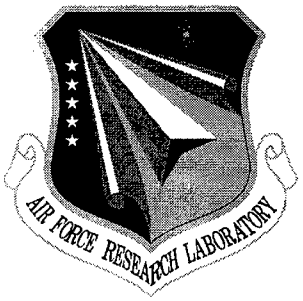
AFRL-IF-RS-TR-2000-154
Final Technical Report
November 2000



INFORMATION INTEGRATION FOR BATTLEFIELD AWARENESS (BADDINFO)

USC Information Science Institute

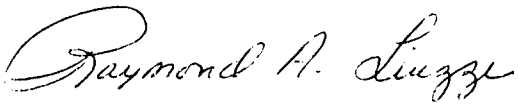
Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. F078

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2000-154 has been reviewed and is approved for publication.

APPROVED: 

RAYMOND A. LIUZZI
Project Engineer

FOR THE DIRECTOR: 

NORTHRUP FOWLER, Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTD, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

INFORMATION INTEGRATION FOR BATTLEFIELD AWARENESS (BADDINFO)

Yigal Arens

Contractor: USC Information Sciences Institute
Contract Number: F30602-97-2-0238
Effective Date of Contract: 30 June 1997
Contract Expiration Date: 16 January 1999
Short Title of Work: Information Integration for Battlefield Awareness (BADDINFO)
Period of Work Covered: Jun 97 - Jan 99

Principal Investigator: Yigal Arens
Phone: (310) 822-1511
AFRL Project Engineer: Raymond Liuzzi
Phone: (315) 330-3577

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Raymond A. Liuzzi, AFRL/IFTD, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	NOVEMBER 2000	Final Jun 97 - Jan 99

4. TITLE AND SUBTITLE
INFORMATION INTEGRATION FOR BATTLEFIELD AWARENESS (BADDINFO)

5. FUNDING NUMBERS
C - F30602-97-2-0238
PE - 63750D
PR - IIST
TA - 00
WU - 15

6. AUTHOR(S)
Yigal Arens

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
USC Information Sciences Institute
4676 Admiralty Way
Marina Del Rey CA 90292-6695

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFTD
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2000-154

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Raymond A. Liuzzi/IFTD/(315) 330-3577

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Information Integration for Battlefield Awareness (BADDInfo) project at USC/ISI had two related components. First, was the development of CSIMS, a C++ version of the SIMS system. Second, the BADDInfo project utilized CSIMS in the development of the Warfighter's Information Packager (WIP), in collaboration with ISX Corporation and Lockheed Martin Intelligent Systems Center.

This final report includes some background that will put the current work in context, which is followed with two parts: A description of the goals of the CSIMS port, and a description of the WIP system. The CSIMS manual and a conference publication describing WIP are attached to the report, as further documentation of the work.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Information Integration, Battlefield Awareness, SIMS, Database, Database Integration, Artificial Intelligence	80
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

1 INTRODUCTION

The Information Integration for Battlefield Awareness (BADDInfo) project at USC/ISI had two related components. First was the development of **CSIMS**, a C++ version of the SIMS system (which was developed in collaboration with the DARPA-funded SIMSPort project at ISI). Second, the BADDInfo project utilized CSIMS in the development of the **Warfighter's Information Packager (WIP)**, in collaboration with ISX Corp. and Lockheed Martin Intelligent Systems Center.

Following some background that will put the current work in context, this report will therefore be divided into two parts: A description of the goals of the CSIMS port, and a description of the WIP system. The CSIMS manual and a conference publication describing WIP are attached to the report, as further documentation of the work.

2 BACKGROUND

2.1 The SIMS Group at ISI

For several years now, with DARPA support, USC/ISI has had a substantial research and prototype development effort in information technology with a particular emphasis on enabling the integration of multiple, distributed heterogeneous sources of data and information. The ISI projects devoted to this pursuit are collectively known as the SIMS group (Single Interface to Multiple Sources).

The SIMS group has developed a modeling system and methodology with which it is possible to declaratively describe the contents, structure and processing capabilities of a variety of types of information sources. We have also developed the SIMS query-processing engine a central query mediator that handles requests for data possibly distributed among multiple sources while insulating the user (or calling system) from the details of database organization, query language, etc. The SIMS mediator accepts queries formulated against a high-level view of the application domain about which data is stored in the multiple sources. It uses the descriptions of the different sources to construct a plan for obtaining the information requested. This query-plan contains steps that involve constructing subqueries, sending such subqueries to appropriate sources, manipulating the results, performing joins, and so forth.

Noteworthy features of the SIMS approach and system are:

- SIMS addresses the "global integration problem"
    - Information sources are mapped into one domain model
    - Information sources are modeled independently of each other
    - The resulting system is highly extensible and maintainable
- SIMS supports flexible and extensible query languages
    - SQL, SQL with path extensions, the Loom KR language
- SIMS dynamically builds query access plans
    - It uses the source descriptions to select relevant information sources and reformulate queries that cannot be answered directly as given
    - It minimizes data movement over the network and maximizes parallelism

    - It interleaves planning and execution for added flexibility
    - It optimizes queries using semantic knowledge
    - It exploits knowledge of both the domain and the information sources
- SIMS handles a variety of information sources
    - Relational databases, object-relational databases
    - Web pages and sites
    - Certain programs

Additional information about SIMS, including downloadable versions of papers describing all aspects of the system, can be obtained at http://www.isi.edu/sims. In particular, see [Arens et al 96] [Arens et al 94] [Arens et al 93] [Knoblock 95].

2.2 SIMS Within the BADD Program

Among the objectives of the BADD program was to ensure, to the extent possible, that all information affecting battlefield decisions be available to the warfighter at all times. In order to achieve this, the capability to assemble the requisite information from the streams of data available from various sources would have to be supported. In addition, the warfighter must have the capability to obtain data that is not resident locally. Limited communication bandwidth, combined with the large size of certain types of data objects (e.g., images), plus the inability to always predict accurately what information could be relevant to the warfighting effort, mean that one cannot always depend on the requisite data being stored locally.

The need to create local proxy-servers for remote data, plus the need to obtain data on the fly from both internal and external sources means, in practical terms, that a Warfighter s Associate (WFA), or another application, must have the ability to identify sources of necessary data, be familiar with the query languages used by various sources, and be capable of querying the sources and integrating the results into an answer suitable for the warfighter s needs. It is necessary that information needs be divided to those that can be satisfied locally and those that require remote access and that the portions of the information obtained from different sources, whether local or remote, be integrated in the end. In addition, information needs that are "ongoing", or repeated at regular intervals, must be coordinated. All this must be done while considering the bandwidth of various network connections, and maintaining the ability to recover if certain sources are found to be inaccessible, or if access is restricted to a degree not anticipated ahead of time.

This is a difficult task. It is unreasonable to expect that an individual would know enough to perform it, and it would be wasteful to build it into every WFA application.

SIMS technology was applied to the BADD domain to provide the capability to integrate information from multiple distributed, heterogeneous battlefield-related information sources. Specifically, (1) SIMS was used to integrate information from multiple data sources to support complex queries specified by data profiles; and (2) the SIMS system, which was previously currently in LISP, was ported to C++, in order to provide an easy integration of SIMS with other BADD components and to help make the SIMS system compliant with the Defense Information Infrastructure Common Operating Environment (DII/COE).

In the next sections, we will discuss the work we carried out to fulfill our technical goals.

3 SIMS PORT TO C++

A port of SIMS to C++ was the foundation of our overall effort of supporting the BADD program. The process was carried out in a series of phases, each of which provided more functionality. The process of porting SIMS to C++ was planned to satisfy the following design goals:

- The resulting C++ system should be efficient, robust and extensible. A phased approach had to be followed to allow for incremental extension of the system s functionality. Each phase had to result in a complete working system.

- Interoperability with the existing Lisp system had to be maintained so that the modules of the original Lisp system could be used in place of the C++ system modules, and vice versa. Maintaining interoperability facilitated smooth transferring of new technologies to the C++ system, and testing the port at various phases to ensure its success.

- The system was designed to facilitate an eventual port to Java. We chose an object-oriented design that would make it relatively easy to transition to Java. We have no current plans for a Java port, but we foresee this as a possible path for future development, due to the multi-platform nature of Java and the strong possibility of its eventual inclusion in DII/COE.

The phased approach was implemented as follows:

	Multi-source access capabilities	**Query language**	**Plan optimization**	**Failure recovery**	**Completion date**
Phase 1	yes	sources must be explicitly specified	no	limited, substitute replicated sources	March,1998
Phase 2	yes	sources must be explicitly specified	yes	full SIMS recovery capabilities	August, 1998
Phase 3	yes	full SIMS language	yes	full SIMS recovery capabilities	December, 1998

3.1 CSIMS System Manual

A complete CSIMS system manual is attached to this report.

4 THE WARFIGHTER S INFORMATION PACKAGER (WIP)[1]

The Warfighters Information Packager (WIP) is a suite of distributed components that create an end-to-end system which allows users to easily obtain information from diverse heterogeneous data sources and display the results in a user-defined, predictable manner.

---

[1] WIP was designed and built together with ISX Corp. and Lockheed Martin Intelligent Systems Center. The BADDInfo project contributed the data access and integration.

WIP tackles the problem of how to satisfy the information needs of individual end-users, and provide access to their specific information given an information push paradigm.

Together, the WIP components create a distributed system that serves as a valuable tool for information analysis in a networked community by:

1. Allowing the user to define high-level information products, "Information Packages", which are parameterized by user interests, needs, and specific tasks and roles;
2. Providing a web-based package viewer that dynamically constructs packages for the user on demand and performs value-added information *linking* based on the information returned for others within the networked community;
3. Allowing users to make high-value complex information requests that can span multiple data sources, without any a priori knowledge of the schema of the sources;
4. Monitoring data sources and *anticipating* useful modifications to a user's Information Package.

4.1 Description

As information research advances, the ability to gather information from diverse sources grows by leaps and bounds. In the case of data however, more isn't necessarily better. Organizing the right information can make the difference between a confusing mass of data and a well organized presentation. This is especially true in the realm of Battlefield Awareness. The Warfighter's Information Packager (WIP) effort is designed to facilitate the accumulation of user requested data and optimize the creation of information packages with regard to display, readability and usability. In this project an intelligent information dissemination system based on user-defined packages, domain models, information visualization techniques, and hyper-linked data technology is being developed.

The WIP system combines the needs of an individual user to gather and analyze information with others within the networked community. One aspect of WIP is the gathering of information by the creation of a high-value complex domain ontology that allows the user access to underlying data-sources without express knowledge of the individual sources themselves. WIP applies technology to anticipate the information needs of a particular user by observing the world-state and detecting changes, and by analyzing the needs of similar users within the system. The query server within WIP includes a query execution planner which decomposes a high-level domain query and dynamically chooses data sources based on their current availability and appropriateness. Once information is gathered WIP again analyzes the acquired information, for a user and for the other users in the system, and recognizes semantic connections between all the users' information in a dynamic fashion. WIP then formats and renders the information product for the user.

4.2 Final Status

Prototype systems, with limited functionality, exist for the Package Editor and Product Viewer. These components provide the user interfaces for the creation of the user information packages and for the viewing of the package products after they have been created. The Query Server (SIMS), responsible for satisfying the information requests, is currently supporting several projects. The Anticipator, which is responsible for anticipating users' information needs, has continued to undergo development by Lockheed Martin.

4.3 Operational Use

The WIP approach provides a straightforward mechanism for multiple levels of command to define their information needs and a convenient way for users to view the results of their information requests. The WIP system addresses information accumulation and analysis issues in the Battlefield Awareness domain. Specifically, the WIP system is being used to create Target Folders that will update automatically based upon available data. Another application currently under development for WIP is providing situation assessments for field operations.

4.4 WIP Description

A complete description of the WIP system in the form of a paper published in the proceedings of the *International Conference Applications of Artificial Intelligence* is attached to this report.

5 REFERENCES

[Arens et al. 96] Yigal Arens, Craig A. Knoblock, and Wei-Min Shen. *Query Reformulation for Dynamic Information Integration*. Journal of Intelligent Information Systems, Vol. 6, 1996, pp. 99-130.

[Arens et al. 94] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, Hoh In and Craig A. Knoblock. Query Processing in an Information Mediator. In *Proceedings of the ARPA/RL Knowledge-Based Planning and Scheduling Initiative Workshop.* Tucson, AZ, February 21—24, 1994.

[Arens et al. 93] Yigal Arens Chin Y. Chee, Chun-Nan Hsu and Craig A. Knoblock. Retrieving and Integrating Data from Multiple Information Sources. *International Journal of Intelligent and Cooperative Information Systems*. Vol. 2, No. 2, 1993. pp. 127—158.

[Knoblock 95] Craig A. Knoblock. *Planning, Executing, Sensing, and Replanning for Information Gathering.* Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Montreal, Canada, 1995.

APPENDIX I: CSIMS MANUAL

The CSIMS Manual, v. 1.0

May 27, 1999

The CSIMS Manual
Version 1.0*

José-Luis Ambite
Yigal Arens
Naveen Ashish
Craig A. Knoblock
Steven Minton
Jay Modi
Maria Muslea
Andrew Philpot
H. Jean Oh
Wei-Min Shen
Sheila Tejada
Weixiong Zhang
Information Sciences Institute and Department of Computer Science
University of Southern California
4676 Admiralty Way,
Marina del Rey, CA 90292, U.S.A.

May 27, 1999

Abstract

SIMS provides intelligent access to heterogeneous, distributed information sources, while insulating human users and application programs from the need to be aware of the location of the sources, their query languages, organization, size, etc.

This manual explains how to bring up a SIMS information server in a new application domain. After providing a short overview of relevant features of the SIMS system, it describes the modeling and programming work that has to be performed to support the extension of SIMS to a given collection of information sources in the domain. To aid a user inexperienced with the technological infrastructure underlying SIMS, the manual contains examples structured as a tutorial that can be followed to actually produce a working SIMS system.

*The research reported here was supported in part by Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency under Contracts Number F30602-94-C-0210, F30602-97-2-0352, and F30602-97-2-0238 and in part by a grant from Computing Devices International. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official opinion or policy of RL, DARPA, the U.S. Government, or any person or agency connected with them.

Contents

1 Introduction

The overall goal of the SIMS project is to provide integrated access to information distributed over multiple, heterogeneous sources: databases, knowledge bases, flat files, Web pages, programs, etc. In providing such access, SIMS tries to insulate human users and application programs from the need to be aware of the location of sources and distribution of queried data over them, individual source query languages, their organization, data model, size, and so forth. The processing of user requests should be robust, capable of recovery from execution-time failures and able to handle and/or report inconsistency and incompleteness of data sources. At the same time SIMS has the goal of making the process of incorporating new sources as simple and automatic as possible.

The SIMS approach to this integration problem has been based largely on research in Artificial Intelligence; primarily in the areas of knowledge representation, planning, and machine learning. A model of the application domain is created, using a knowledge representation system to establish a fixed vocabulary for describing objects in the domain, their attributes and relationships among them. Using this vocabulary, a description is created for each information source. Each description indicates the data-model used by the source, the query language, network location, size estimates, etc., and describes the contents of its fields in relation to the domain model. SIMS' descriptions of different information sources are independent of each other, greatly easing the process of extending the system. Some of the modeling is aided by source analysis software developed as part of the SIMS effort.

Queries to SIMS are written in a high-level language (Loom or a subset of SQL) using the terminology of the domain model — independent of the specifics of the information sources. Queries need not contain information indicating which sources are relevant to their execution or where they are located. Queries do not need to state how information present in different sources should be joined or otherwise combined or manipulated.

SIMS uses a planner to determine how to identify and combine the data necessary to process a query. In a pre-processing stage, all data sources possibly relevant to the query are identified. The planner then selects a set of sources that contain the queried information and generates an initial plan for the query. This plan is repeatedly refined and optimized until it meets given performance criteria. The plan itself includes, naturally, sub-queries to appropriate information sources, specification of locations for processing intermediate data, and parallel branches when appropriate. The SIMS system then executes the plan. The plan's execution is monitored and replanning is initiated if its performance meets with difficulties such as unexpectedly unavailable sources. It is also possible for the plan to include explicit replanning steps, after reaching a state where more is known about the circumstances of plan execution.

Changes to information sources are handled by changing source descriptions only. The changes will automatically be considered by the SIMS planner in producing future plans that utilize information from the modified sources. This greatly facilitates extensibility.

The rest of this section presents an overview of SIMS and its architecture. In Section 2 we show the format of the queries that a user would input to SIMS and the output that should be expected. Then we consider in more detail the specification of the domain model, in Section 3, and how information sources are described to the system, in Section 4. Section 8 gives a brief introduction on how to construct a wrapper for a new information source and how to communicate with the wrapper. Section 10 explains how to run SIMS both through its graphical user interface and its functional interface. Section ?? describes how to test and debug a new SIMS application. Section 11 presents the installation and system requirements. Finally, in Section 12 we show the code that would implement the example that is discussed throughout the manual. Section 13 contains a reading list of relevant papers.

1.1 Architecture and Background

A visual representation of the components of CSIMS is provided in Figure 1.

CSIMS addresses the problems that arise when one tries to provide a user familiar only with the general domain with access to a system composed of numerous separate data- and knowledge-bases.

Specifically, CSIMS does the following:

- **Modeling:** It provides a consistent way of describing information sources to the system, so that data

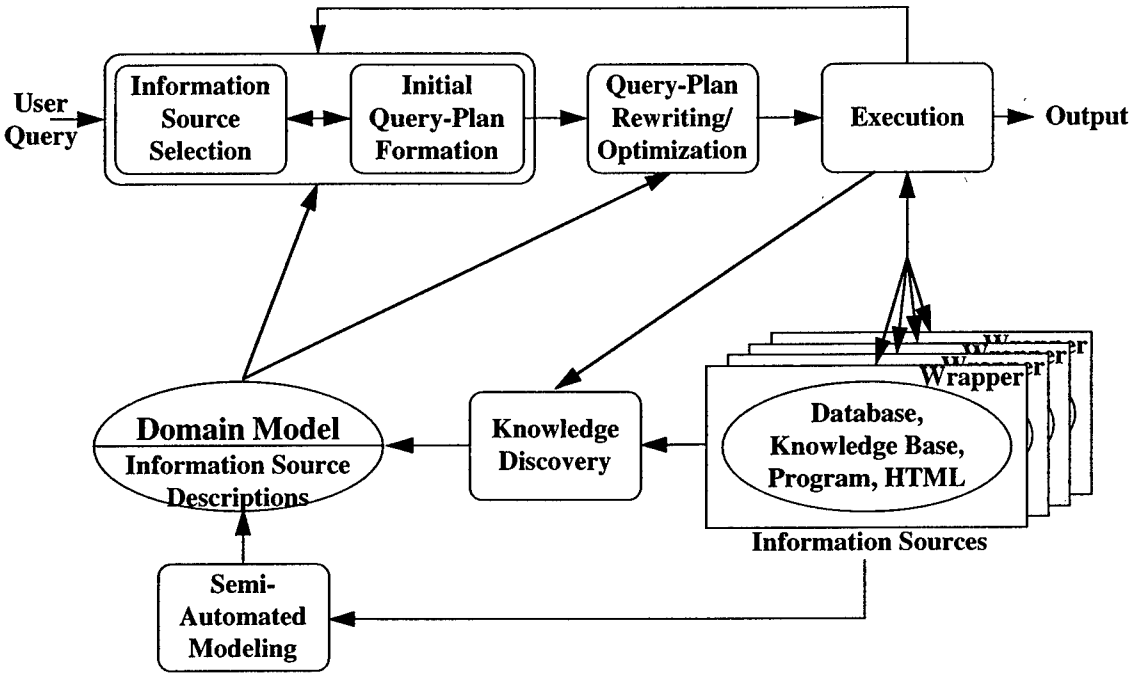


Figure 1: CSIMS Overview Diagram.

in them is accessible to it (currently not implemented in CSIMS).

- **Information Source Selection:** Given a query, it
  - Determines which classes of information will be relevant to answering the query.
  - Quickly, using some information generated during an earlier preprocessing stage, generates a list of all combinations of sources that contain all information required for a query.
- **Initial Query-Plan Formation:** It creates an initial plan, a sequence of subqueries and other forms of data-manipulation that when executed will yield the desired information. This initial plan does not necessarily satisfy any optimization requirements.
- **Query-Plan Rewriting/Optimization:** By successively applying rewriting rules that preserve the correctness of the plan, it gradually improves the plans efficiency. This process continues until no further rewriting is possibly, or until the allotted time runs out.
- **Execution:** It executes the reformulated query plan; establishing network connections with the appropriate information sources, transmitting queries to them and obtaining the results for further processing. During the execution process CSIMS may detect that certain information sources are not available, or respond erroneously. In such cases, the relevant portion of the query plan will be replanned.

Each information source is accessed through a *wrapper*, a module that can translate from a description of a set of data in CSIMS' internal representation language into a query for that data that is then submitted to the source. The wrapper also handles communication with the information source and takes the data returned by it and sends it on to CSIMS in the form CSIMS expects.

1.2 Information Sources Supported

In order for CSIMS to support an information source it must have a description of the source, and there must exist a wrapper for that type of source. While each information source needs to be described individually, only one wrapper is required for any type of information source.

In addition, through an "ODBC wrapper" CSIMS uses ODBC to interact with all ODBC-enabled databases. This includes Oracle, Sybase, Informix, Ingres, and many others. To add a new database of any of these types requires, therefore, only to create an information source description for it. In order to add an information source of a new type one would have to obtain, or write, a new wrapper for it as well. We also have an ongoing associated effort (Ariadne) that includes work on semi-automatic generation of wrappers for HTML pages.

2 The CSIMS Query Language

Currently, CSIMS only supports commands for retrieving data. Specifically, CSIMS takes a retrieval query as input and returns the data satisfying the constraints specified in the query. The output format of CSIMS is a list of tuples of constant(s). A retrieval query can be expressed in a LOOM syntax or a SQL syntax. The following two sections discuss these two languages in detail.

2.1 LOOM Syntax

Loom serves as the knowledge representation system that CSIMS uses to describe the domain model and the contents of the information sources. In addition, Loom is used to define a knowledge base that itself serves as an information source to CSIMS. Loom provides both a language and an environment for constructing intelligent applications. It combines features of both frame-based and semantic network languages, and provides some reasoning facilities.

The BNF syntax for the CSIMS query language is shown in Figure 2.

```
<query> ::= (sims-retrieve <variable> | ({<variable>}+) <query-expr>)
<query-expr> ::= ({:and | :or} {<query-expr>}+)
<clause> ::= <concept-exp> | <relation-exp> | <assignment-exp> | <comparison-exp>
<concept-exp> ::= (<concept-name> <variable>)
<relation-exp> ::= (<relation-name> {<bound-variable>} {<term>})
<assignment-exp> ::= (:= <unbound-variable> {<arith-exp> | <set-exp>})
<set-exp> ::= ({<constant>}+)
<comparison-exp> ::= <member-comparison> | <arithmetic-comparison>
<member-comparison> ::= (member <bound-variable> <set-exp>))
<arithmetic-comparison> ::= (<comparison-op> {<arith-exp>} {<arith-exp>})
<arith-exp> ::= <number> | <bound-variable> | (<arith-op> <arith-exp> <arith-exp>)
<arith-op> ::= + | - | * | /
<comparison-op> ::= = | > | < | >= | <= | != | match
<concept-name> ::= <symbol>
<relation-name> ::= <symbol>
<term> ::= <constant> | <variable>
<variable> ::= <bound-variable> | <unbound-variable>
<bound-variable> ::= ?<symbol>
<unbound-variable> ::= ?<symbol>
<constant> ::= <number> | <string>
```

Figure 2: BNF for the CSIMS Query Language in LOOM Syntax

The following are the basic forms of a CSIMS query:

`(sims-retrieve` $?v$ `<query-expr>)`

`(sims-retrieve (`$?v_1$ ... $?v_n$`) <query-expr>)`

The variables listed after the `sims-retrieve` command, $?v$ and $?v_1 \ldots ?v_n$, are considered output variables. This means that the values of these variables are returned as the output of the query. All variables must be named with the prefix '?'. The query expression is composed of clauses and constructors. Clauses determine the values of the variables by binding the variables to specific types of values. In other words, clauses constrain the values of the variables. There are four types of clauses supported by the CSIMS language which will be described in the next section. Clauses can be grouped by constructors into queries. Currently, the constructors provided are `:and` and `:or`.

A CSIMS query returns as output a list of instantiations of the output variables which satisfy the bindings of the clauses in the query body. The following shows an example of output from a CSIMS query.

```
(sims-retrieve (?name)  (:and (American-Large-Seaport ?seaport)
                              (port-name ?seaport ?name)))
```

```
==> (("Long Beach") ("New York") ("Norfolk") ...)
```

In this query the output variable ?name is bound to the values of the role port-name of American-Large-Seaport.

2.1.1 Clauses

Clauses are expressions that constrain the values which can be bound to a variable. A clause is satisfied when there exists values that satisfy the constraints on the variables in that clause. The following are the four types of clauses:

- Concept expressions:

  ```
  (<concept-name> <variable>)
  ```

  where <concept-name> is the name of a concept, the variable is bound to an instance of the concept <concept-name>. An example of a concept expression is:

  ```
  (Seaport ?seaport)
  ```

  This constrains the variable ?seaport to only the instances of the concept Seaport. Variables in concept expressions cannot be returned by the system.

- Relation expressions:

  ```
  (<relation-name> <bound-variable> <term>)
  ```

  where <relation-name> is the name of a relation, <bound-variable> is a variable from a concept expression while <term> can be either a variable or a constant (a number or a string). The first clause states that there is a binary relation <relation-name> between <bound-variable> and <term>. The following are examples of this type of relation expression:

  ```
  (port-name ?seaport ?name)
  (seaport-country-code ?portCountryCode 'A123)
  ```

  The first expression is only satisfied if the value for ?name is the port-name of ?seaport. The second expression is only satisfied if 'A123 is the seaport-country-code of ?portCountryCode.

- Assignment expressions:

  ```
  (:= <unbound-variable> <arith-expr>)
  ```

  This clause assigns to the unbound variable the computed result of <arith-expr>.

  For the following example, suppose we have a concept Seaport and its relations to its name (port-name) and to its number of cranes code (cranes). The following query will return a list of the names of a pair of seaports that have more than five cranes in total.

  ```
  (sims-retrieve (?portname1 ?portname2)
                 (:and (Seaport ?seaport1)
                       (Seaport ?seaport2)
                       (port-name ?seaport1 ?portname1)
                       (port-name ?seaport2 ?portname2)
                       (cranes ?seaport1 ?cranes1)
                       (cranes ?seaport2 ?cranes2)
                       (:= ?totalcranes (+ ?cranes1 ?cranes2))
                       (> ?totalcranes 5)))
    ==> (("Long Beach" "Norfolk")
         ("New York" "San Diego")
         :
        )
  ```

- Comparison expressions are used to express a constraint on variables. The following are forms of member comparisons:

```
(member <bound-variable> <set-exp>)
```

where a <set-exp> is defined as a set of constants. This clause is satisfied if the variable is bound to one of the constants (i.e., strings or numbers) in the <set-exp>.

The following are examples of member comparisons:

```
(member ?name ("Long Beach" "San Diego" "Newport Beach"))
```

This expression is only satisfied if the value for `?name` matches one of the three strings in the set.

Another type of comparison expression uses the arithmetic comparison operators: `=`, `>`, `<`, `>=`, `<=`, `!=`.

```
(<comparison-op> <arith-expr> <arith-expr>)
```

The following are examples of the arithmetic comparison:

```
(> ?cr 5)
(= ?depth 120)
```

The first example checks that the the number of cranes (?cr) of a seaport is greater than five. The second example verifies the channel depth (?depth) of a seaport is equal to 120.

`Match` is yet another comparison expression. It takes two arguments, a variable and a match string. It matches the value of the variable against the string. The string can have two meta-characters. The first meta-character is %, which matches zero or more characters. The second is _, which matches any one character. The following is an example of using `match`:

```
(sims-retrieve (?name)
   (:and (geographic-location ?g)
         (geographic-name ?g ?name)
         (match ?name "_X%Y%")))
```

This query will retrieve all geographic locations whose names have `X` as the second character and at least one `Y` after the `X`.

2.1.2 Query Expression Constructors

This section describes the two expression constructors supported by CSIMS.

`(:and` $expr_1$ ... $expr_n$`)` — CONJUNCTION

This returns the values for which each of the expressions $expr_j$ is satisfied.

Example: `(:and (Seaport ?x) (port-name ?x ?y))`
This expression is satisfied if `?x` is a `Seaport` and `?y` is the name of that seaport.

`(:or` $expr_1$ ... $expr_n$`)` — DISJUNCTION

This returns the values for which at least one of the expressions $expr_j$ is satisfied.

Example: `(:or (Small-Seaport ?x) (American-Large-Seaport ?x))`
This expression is satisfied if `?x` is either a `Small-Seaport` or an `American-Large-Seaport`.

2.2 SQL Syntax

CSIMS also accommodates queries written in a subset of SQL syntax. A query in SQL syntax is first translated into the native Loom query language and then processed by CSIMS internally. This SQL-syntax front end is different from a typical SQL query engine, such as a relational database, in two important ways.

- Syntax: CSIMS' SQL front end accepts only a subset of standard SQL, a subset which easily corresponds to the internal Loom query language variant used in CSIMS.

- Semantics: CSIMS' SQL front end uses SQL to refer to CSIMS domain concepts and relations, which are high level source-independent descriptions ("views") of the application domain. The terms do not necessarily refer to tables in any particular database.

The BNF syntax for the CSIMS query language is shown in Figure 3.

```
<query> ::= SELECT <ret-param>{,<ret-param>}+
            FROM <concept-spec>{, <concept-spec>}+
            WHERE condition {, condition}+
<ret-param> ::= <colname> | <expr>
<colname> ::= <CONCEPT-NAME>.<ATTRIBUTE-NAME> | <ALIAS>.<ATTRIBUTE-NAME>
<concept-spec> ::= <CONCEPT-NAME> | <CONCEPT-NAME> <ALIAS>
<expr> ::= (<expr> {,expr}+) |
           <constant> |
           -<expr> | +<expr> |
           <expr> <op> <expr>
<constant> ::= <NUMBER> | <STRING> | NULL
<op> ::= * | + | - | /
<comparison ops> ::= = | != | <> | > | < | >= | =< | match
<condition> ::= <expr> <comparison-ops> <expr> |
                <expr> IN <expr> |
                <expr> LIKE <expr> |
                <condition> AND | OR <condition> |
                NOT <condition> |
                (<condition>)
```

Figure 3: BNF for the CSIMS Query Language, SQL Syntax

In both SELECT lists and constraint conditions, attributes must always be specified using the fully qualified (`Concept.attribute`) syntax, even if only a single concept is referenced. This is because parsing of the SQL might take place in an environment where the schema of the underlying view might not be available, so there might be no context providing a way to assign attributes to concepts. The following is a correct example:

```
SELECT ConceptZ.a
FROM ConceptZ
WHERE ConceptZ.b > 10
```

while the next two examples are incorrect because attributes are not specified with the fully qualified syntax.

```
SELECT a
FROM ConceptZ

SELECT ConceptW.b
FROM ConceptW
WHERE b like "%LARGE%"
```

As alluded to above, aliases can be used if desired:

```
SELECT R.a, R.b, S.b, S.c
FROM ConceptX R, ConceptY S
WHERE R.d = S.e
```

However, `Concept.*` (meaning all attributes) is not supported. In addition, CSIMS does not currently distinguish between sets and bags of tuples. Practically, this means that in SELECT statements everything is distinct.

The following is an example of CSIMS query in SQL format

```
SELECT Seaport.port-name
FROM Seaport
WHERE Seaport.cranes > 7
```

```
==> ("Long Beach" "Norfolk" ...)
```

This query asks for the names of large seaports. Equivalently, the following query returns the same information.

```
SELECT Large-Seaport.port-name
FROM Large-Seaport
```

There are a few exceptions in CSIMS' SQL support. Constraints and expressions are currently only expressed in terms of simple arithmetic and boolean operators. In addition, CSIMS' treatment of aggregate operations is currently very limited. Specifically, the following are not supported:

- Nested SELECTs.
- START WITH, GROUP BY, HAVING, CONNECT BY conditions.
- ORDER BY, FOR UPDATE.
- set operations UNION, UNION ALL, INTERSECT, MINUS.
- use of ROWNUM or other pseudocolumns.

Finally, as a (Lisp) syntactic convenience, the system is configured by default to intepret any occurrence of the underscore character (_) in concept names and attribute names to the dash (-) character. For example, the concept-name RUNWAY_LENGTH would be rendered as RUNWAY-LENGTH. This detail is unimportant except for users attempting to match up with a preexisting CSIMS domain model using terms denoted with the (-) character.

3 The Domain Model

A domain model provides the general terminology for a particular application domain. This model is used to unify the various information sources that are available and provide the terminology for accessing those information sources. Throughout this manual we use a simple application domain that involves information about various types of seaports. The example is simple so that we can provide a complete, but short, description of the model. Figure 4 shows our example domain model.

Note: Currently CSIMS does not implement the domain model.

3.1 The Model: Classes and Attributes

The domain model is described in the Loom language, which is a member of the KL-ONE family of KR systems. In Loom, objects in the world are grouped into "classes". Our example domain has several classes: Seaport, Large Seaport, Small Seaport, American Large Seaport, European Large Seaport and Country. The classes are indicated with circles in Figure 4. *Subclass* relationships are shown by dark solid arrows. For example, the class `Large Seaport` is a subclass of `Seaport`. This means that every instance of `Large Seaport` is also an instance of `Seaport`. A class can have any number of subclasses, but (currently) CSIMS allows a class to have at most one superclass.

The figure also shows that `Large Seaport` and `Small Seaport` form a *covering*. This means that the class `Seaport` is the union of `Large Seaport` and `Small Seaport`. Thus, every seaport is either a large seaport or a small seaport.

Classes generally have *attributes* associated with them. For instance, the class `Seaports` has six attributes associated with it, a geographic code (`geoloc-code`), a port name (`port-name`), the number of cranes in the port (`cranes`), the channel depth (`depth`), the seaport's country code (`seaport-country-code`), and a country (`country`). This means that every seaport has a corresponding geographic code, port name, number of cranes, depth, and country code. Attributes are *inherited* down to subclasses. Thus, every large seaport will also have these six attributes, since `Large Seaport` is a subclass of `Seaport`. European large seaports have seven attributes, since they inherit the six attributes from `Large Seaport`, plus there is an additional attribute, the `tariff code`, associated with that class.

Classes can be defined as either primitive classes or they can be defined in terms of other classes. A primitive class has no explicit definition specifying the constraints that differentiate it from its superclass. For example, one might could create the class `Large Seaport` without specifying what constraints differentiate it from its superclass, `Seaport`. In terms of modeling a set of sources, this is useful in the case where you have two sources, where one is clearly a subclass of the other, but there is no simple way to characterize the specific subclass of information it contains.

Alternatively, it is possible to define the relationship between a subclass and superclass by explicitly describing the constraints on the subclass. For example, a large seaport might be defined as a seaport with more than seven cranes. This is what we have done in our example domain, as shown in Figure 4, where the class `Large Seaport` is defined as `Seaport` $\wedge$ `(> cranes 7)`.

3.2 Specifying the Model: Class Definitions

Figure 5 shows the six class definitions that must be given to CSIMS to specify the classes in our example domain. (See Figures 7 and 8 for a BNF description of the modeling language.) The class definition indicates whether or not the class is primitive. When a class is defined in terms of other classes, such as `Large Seaport`, the definition is specified using an "is" clause. The "is" and the "is-primitive" clauses are also used to indicate any attributes associated with the class.

Class definitions also have an "annotations" field. CSIMS requires that every class has at least one defined *key*, which consists of one or more attributes that uniquely identify each instance in a class. Since there may be more than one way to uniquely identify an instance, a class can have multiple keys. For example, any seaport can be uniquely identified either using the `geoloc-code` or the `port-name`. Because more than one attribute may be necessary to uniquely identify an instance, a key can include multiple attributes. For instance, in another domain, it might be that street number, street name and city name are all necessary to uniquely identify a particular house.

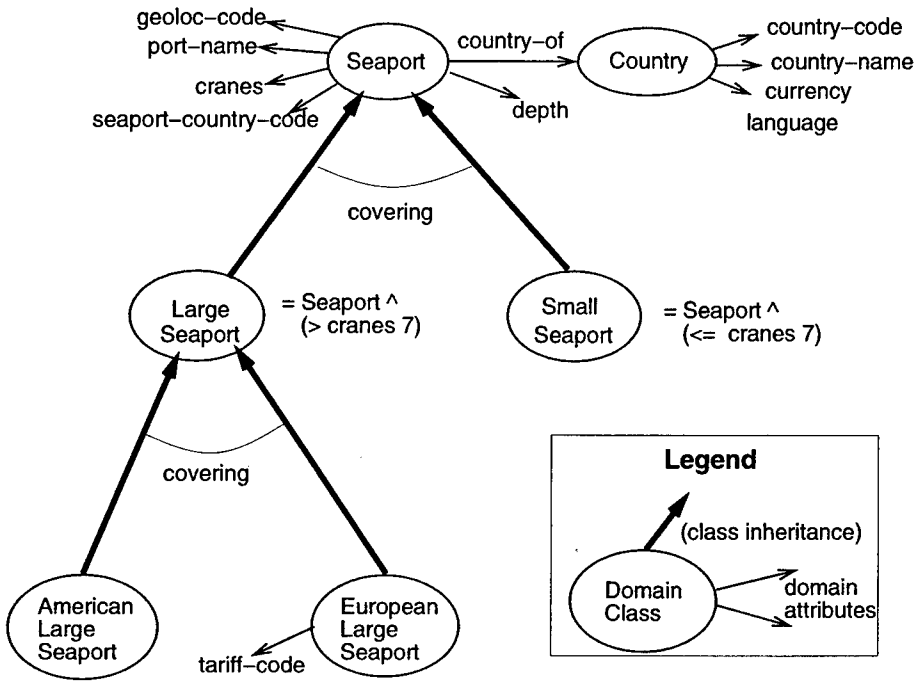


Figure 4: Domain Model

The annotations field of a class definition is also used to indicate that a class is a covering (i.e., the union) of some of its subclasses. For example, the class `Seaport` is the union of `Large Seaport` and `Small Seaport`.

3.3 Specifying the Model: Attribute Definitions

There are two types of attributes in CSIMS. Most of the attributes in our example domain are *simple attributes*, in that they are basic classes: strings or numbers. But attributes can also represent relations between two defined classes. For instance, `Seaport` has an attribute called `Country-of`, so that every seaport is associated with a country. Thus, `Country-of` is a relation between `Seaport` and `Country`.

Figure 6 shows the relation definitions that define the attributes used in the domain model. Notice that each attribute has a domain and range. Defined relations (attributes that relate two classes) have a definition. For instance the `Country-of` relation has a definition which specifies that the relation holds between a seaport and a country if the seaport's `seaport-country-code` matches the country's `country-code`.

There cannot be two different attributes with the same name. In our example, `Seaport` has the attribute `seaport-country-code`, and `Country` has an attribute `country-code`. Even though these are both 'country codes', the names of the attributes must be different.[1]

The domain model is used as the basis for the CSIMS query language that enables the user to construct queries. The classes included in the domain model are not necessarily meant to correspond directly to objects described in any particular information source. The domain model is intended to be a description of the application domain from the point of view of someone who needs to perform real-world tasks in that domain

[1]Had we wanted a `Seaport` to have an attribute called `country-code`, we would have done so only by making the model more complex. For instance, we could have created a class `Geographic entity` with an attribute `country-code`, which could have been a superclass of both `Country` and `Seaport`, in which case the attribute `country-code` would have been inherited down to both of these classes.

```
(def-sims-concept seaport
    :is-primitive (:and sims-domain-concept
                        (:the country-of country)
                        (:the geoloc-code string)
                        (:the seaport-country-code string)
                        (:the port-name string)
                        (:the cranes number)
                        (:the depth number))
    :annotations ((key (geoloc-code))
                  (key (port-name))
                  (covering (large-seaport small-seaport))))

(def-sims-concept large-seaport
    :is (:and seaport
              (> cranes 7))
    :annotations ((key (geoloc-code))
                  (key (port-name))
                  (covering (american-large-seaport
                             european-large-seaport))))

(def-sims-concept small-seaport
    :is (:and seaport
              (<= cranes 7))
    :annotations ((key (geoloc-code))
                  (key (port-name))))

(def-sims-concept american-large-seaport
    :is-primitive large-seaport
    :annotations ((key (geoloc-code))
                  (key (port-name))))

(def-sims-concept european-large-seaport
    :is-primitive (:and large-seaport
                        (:the tariff-code string))
    :annotations ((key (geoloc-code))
                  (key (port-name))))

(def-sims-concept country
  :is-primitive (:and sims-domain-concept
                      (:the country-code string)
                      (:the country-name string)
                      (:the currency string)
                      (:the language string))
  :annotations ((key (country-code))))
```

Figure 5: Class Definitions for our Example Domain

and/or to obtain information about it. CSIMS is designed to allow users to query the domain model without specific knowledge of the way the actual information sources relate to the domain model. The next section describes how application developers describe the actual information sources and their relationship to the

domain model.

```
;;; Seaport attributes

(def-sims-relation geoloc-code
    :domain seaport
    :range string)

(def-sims-relation port-name
    :domain seaport
    :range string)

(def-sims-relation cranes
    :domain seaport
    :range number)

(def-sims-relation depth
    :domain seaport
    :range number)

(def-sims-relation seaport-country-code
    :domain seaport
    :range string)

(def-sims-relation country-of
    :domain seaport
    :range country
    :is (:satisfies (?s ?c)
          (:and (seaport ?s)
                (country ?c)
                (seaport-country-code ?s ?cc)
                (country-code ?c ?cc))))


;;; European Large Seaport attributes

(def-sims-relation tariff-code
    :domain european-large-seaport
    :range string)


;;; Country attributes

(def-sims-relation country-code
    :domain country
    :range string)

(def-sims-relation country-name
    :domain country
    :range string)

(def-sims-relation currency
    :domain country
    :range number)

(def-sims-relation lang
    :domain country
    :range number)
```


Figure 6: Attribute Definitions for our Example Domain

```
class-definition ::=
   (DEF-SIMS-CONCEPT ClassName
        is-clause
        annotations-clause)

is-clause ::=
   :IS-PRIMITIVE (:AND SuperClassName attr-clause*)  |
   :IS (:AND SuperClassName constraint-expr*)

annotations-clause ::=
   :ANNOTATIONS (annotation+)

annotation ::=
   (KEY (AttributeName+)) |
   (COVERING (SubClassName SubClassName+))

attr-clause ::=
   (:THE AttributeName ClassName) |
   (:THE AttributeName STRING) |
   (:THE AttributeName NUMBER)

constraint-expr ::=
   (test Term Term) |
   (:FILLED-BY AttributeName Term) |
   (:NOT-FILLED-BY AttributeName Term)

test ::=
   >  |  <  |  >=  |  <=  |  !=  |
```

Figure 7: BNF for Class Definitions

```
simple-relation ::=
  (DEF-SIMS-RELATION RelationName
     :DOMAIN ClassName
     :RANGE [NUMBER | STRING])

defined-relation ::=
  (DEFRELATION RelationName
     :DOMAIN ClassName
     :RANGE ClassName
     :IS (:SATISFIES (VariableName VariableName) constraint-expr))

constraint-expr ::=
   (:FOR-SOME (VariableName) constraint-expr)
   (:AND constraint-expr+) |
   (AttributeName Term Term) |
   (ConceptName Term) |
   (test Term Term)

test ::=
   >  |  <  |  >=  |  <=  |  !=  |
```

Figure 8: BNF for Attribute Definitions

4 Defining Information Sources

In order to extract and integrate data from an information source, a person building an application must describe the contents of the source using terms from the domain model and define the details of how the source is accessed. Each of these issues is addressed in turn.

4.1 Describing the Contents of an Information Source

Each information source is incorporated into CSIMS by describing the data provided by that source in terms of the domain model presented in the previous section. This description provides the following information:

- The precise class of instances provided by a source.
- The set of attributes that are available from the source.
- The name of the source that provides the data (the next section will define additional information about accessing each source)
- The mapping from the table/class name of the source and the name used in the domain model.
- The mapping from the attribute names used in the source and those used in the domain model.

To illustrate the principles involved in representing an information source within CSIMS, consider how a set of sources would be represented using the domain model described in the previous section. Figure 9 shows the example domain model linked to a set of seven separate sources.

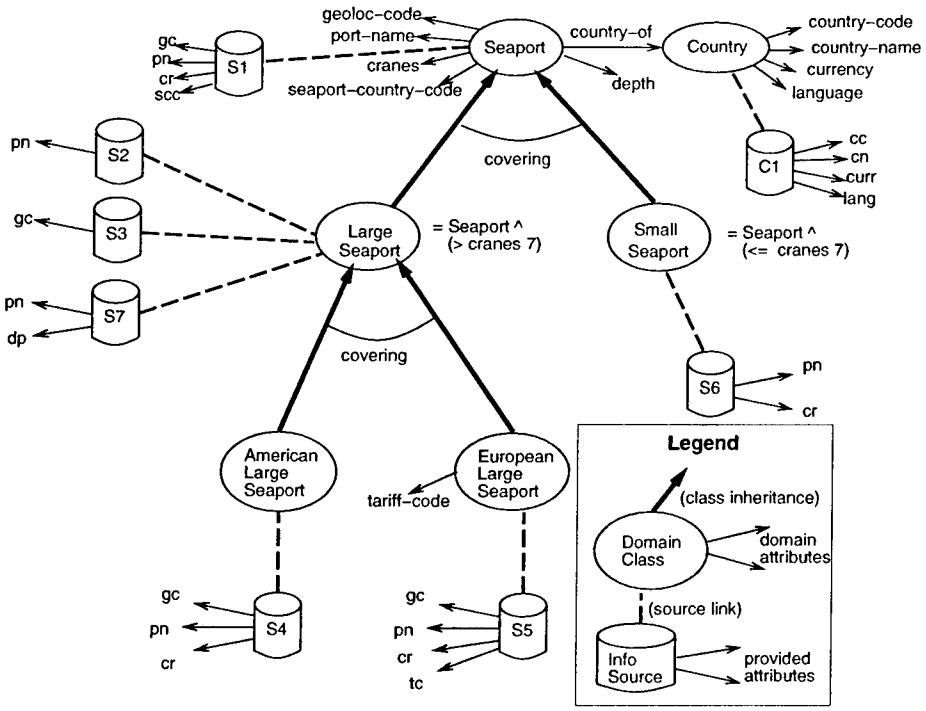


Figure 9: A Set of Sources Described by a Domain Model

In the figure, each source is linked to a class with a dashed line. The meaning of such a link is that the source provides exactly the set of instances described by the class of the domain model. Thus, the figure shows that S2, S3, and S7 all provide *exactly* the same set of large seaports. If there is another source that provides only a subset of a class of instances, then a new subclass in the domain model would be created

and the source would be linked to that class. Sources S4 and S5 are both examples of sources that provide subclasses of large seaports and thus are linked to the appropriate subclasses in the domain model.

Since different sources often provide different attributes for the same class, we do not require that all sources provide all attributes of a class. In the figure, the attributes provided by each source are shown next to the individual sources.

The general form of a source description statement is shown in Figure 10. There will generally be one of these statements for each table or relation in a source. However, in some cases, sources can be more naturally modeled by mapping a single relation in the source into more than one domain class. As shown in the figure, a domain class is used to describe a source table and DB and the domain attributes are linked to the corresponding attributes of the source.

```
(source-description <domain-class> <source-table> <source-db>
    (<domain-attribute-1> <source-attribute-1>)
    (<domain-attribute-2> <source-attribute-2>)
    ...
    (<domain-attribute-n> <source-attribute-n>))
```

Figure 10: General Form of a Source Description

Consider how a specific source in Figure 9 would be described. For source `S7`, which provides the port name and depth of `Large-Seaport`. The description of this source is shown in Figure 11.

```
(source-description Large-Seaport S7 EXKB7
    (port-name pn)
    (depth dp))
```

Figure 11: Source Description for Large-Seaports table of S7 Database

5 Accessing an Information Source

In addition to specifying the content of an information source, the system also needs to know *what* information sources are currently available and *how* to access them. This section describes the basic commands for declaring information sources.

To make a source available to the system, certain information about the source must be declared in advance. This information is provided in a IS_FILE (information source file). Currently there are four predefined source types, and they are explained in the subsections below.

5.1 kqml-odbc-source

A *kqml-odbc-source* is a source that supports ODBC communication and the full SQL query language. We provide KQML-based wrappers that allow CSIMS to communicate with most of the commericial relational database systems (for details see 8 and 9.1). An example of how such a source would be declared is shown below:

```
(source-type source-name host-name agent-name db-name userid source-status)
```

where:

source-type defines the specific type of source.

source-name provides the name of the information source within CSIMS.

host-name is the name of the machine on which the information source is running.

The *agent-name* of a source is the unique name that KQML uses to identify the wrapper of that particular source.

The *db-name* is the internal database name, which might *not* be unique since one may have multiple instances of the same information source running on different hosts.

The *userid* is the internal `userid` for a database.

source-status defines the status of a source (can be either `UP` or `DOWN`).

Suppose that source S2 from the previous example (Figure 9) is an Oracle database (i.e., it supports ODBC communication) named "assetss" that can be accessed with the userid "abc". It is located on host "isd54.isi.edu" and the wrapper for this source is called "sql_server". We declare this source as follows:

```
(KQML_ODBC_SOURCE "S2" "isd54.isi.edu" "sql_server" "assetss" "abc" UP)
```

5.2 kqml-wrapper-source

A *kqml-wrapper-source* is a wrapper that makes the web sources look like a database and that communicates through KQML interface.

An example of how such a source would be declared is shown below:

```
(source-type source-name host-name agent-name source-status)
```

where:

source-type defines the specific type of source.

source-name provides the name of the information source within CSIMS.

host-name is the name of the machine on which the information source is running.

The *agent-name* of a source is the unique name that KQML uses to identify the wrapper of that particular source.

source-status defines the status of a source (can be either `UP` or `DOWN`).

Suppose that source S3 from the previous example (Figure 9) is a wrapper for a certain website (i.e. www.lg-seaport.com). Since we access this wrapper through KQML, what we need to know are its KQML agent name and the host name where it is located.

```
(KQML_ODBC_SOURCE "S3" "isd54.isi.edu" "lg-seaportwrapper" UP)
```

5.3 http-wrapper-source

A *http-wrapper-source* is a wrapper that runs through CGI. CSIMS communicates with this type of sources via HTTP.

An example of how such a source would be declared is shown below:

```
(source-type domain-name wrapper-name host-name cgi-path port source-status)
```

where:

source-type defines the specific type of source.

domain-name provides the name of the domain name within CSIMS.

wrapper-name provides the name of the wrapper information source within CSIMS.

host-name is the host name of the machine on which the information source is running.

cgi-path is the relative path to where wrapper.cgi resides.

port is the port number that is used for the HTTP server (default : 80).

source-status defines the status of a source (can be either `UP` or `DOWN`).

Suppose that source S4 from the previous example (Figure 9) is a http-wrapper for a certain website (i.e. www.lg-seaport.com).

```
(HTTP_WRAPPER_SOURCE "seaport-domain" "S4" "isd56.isi.edu" "/cgi-bin/" 8080 UP)
```

5.4 functional-source

A *functional-source* is a special type of source that is considered to have *in-attributes* and *out-attributes*. Given a set of in-attributes, the functional source provides the corresponding out-attributes. Each functional source is represented as a user defined function that takes as arguments the in-attributes and produces the out-attributes. A user that wants to add a functional source must provide both the source declaration in the IS_FILE and a user-defined function that implements the source. An example of how such a source would be declared is shown below:

```
(source-type source-id line-id source-name source-function source-status)
(source-type source-id line-id In-Attr1 In-Attr2 ...)
(source-type source-id line-id Out-Attr1 Out-Attr2 ...)
```

where:

source-type defines the specific type of source.

source-id is a unique functional source ID.

line-id describes the type of information provided by the current line of the functional source description. It can take the values INFO, IN_ATTR and OUT_ATTR.

source-name provides the name of the information source within CSIMS.

source-function represents the function name of this specific functional source.

source-status defines the status of a source, that can be UP or DOWN.

In the example presented in Figure 9, source "C2" is a functional source that given the country-border length in kilometers provides the corresponding length in miles. If the user-defined function corresponding to this source is called "KmsToMiles", we would declare this source as:

```
(FUNCTIONAL_SOURCE 1 INFO "C2" "KmsToMiles" UP)
(FUNCTIONAL_SOURCE 1 IN_ATTR "km")
(FUNCTIONAL_SOURCE 1 OUT_ATTR "mi")
```

6 The CSIMS Axiom Language

CSIMS uses axioms to determine the set of information sources that can provide the necessary information to answer a specific query. Currently, CSIMS accepts axioms that are provided in the format that we will describe below. Axioms can be provided by RISCSIMS, or they can be written by hand. Given a domain model (see Section 3) and a set of information source definitions (see Section 4), RISCSIMS is a system that produces the relevant axioms for the specified domain. If RISCSIMS is not available, axioms can be written by hand.

6.1 Axiom Syntax

Before defining the actual axioms one has to provide first a *domain schema* and a *source schema.*

6.1.1 Domain Schema

For each concept definition in the domain model (see Figure 5) there is a corresponding definition in the domain schema. In the domain schema, all concept names and attribute names represent domain terms. A concept definition in the domain schema is represented as:

```
domain_concept(domain_attribute-1 domain_attribute-2 ... domain_attribute-n)
```

where:
<domain_concept> := <string> , the domain concept name
<domain_attribute> := <string> , the domain attribute name
Looking at the class definitions for our example domain (Figure 5) we write the following domain schema.

```
country(country_code country_name currency kilometers language miles)
european_large_seaport(country_of cranes depth geoloc_code port_name seaport_country_code tariff_code)
american_large_seaport(country_of cranes depth geoloc_code port_name seaport_country_code)
large_seaport(country_of cranes depth geoloc_code port_name seaport_country_code)
small_seaport(country_of cranes depth geoloc_code port_name seaport_country_code)
seaport(country_of cranes depth geoloc_code port_name seaport_country_code)
```

6.1.2 Source Schema

For each source in the set of sources described by a domain model (Figure 11) there is a corresponding definition in the source schema. In the source schema, concept names and attribute names represent source terms. A source definition in the source schema is represented as:

```
source_concept(source_attribute-1 source_attribute-2 ... source_attribute-n)
```

where:
<source_concept> := <source_concept_name>%<table_name>
<source_concept_name> := <string>, the name of the source
<table_name> := <string>, the name of the table within the source
<source_attribute> := <source_attribute_name> | <constant> | <sql_function>
<source_attribute_name> := <string> | $<string>
<constant> := <string> | <number>
<sql_function> := ...

- String constants are enclosed with double quotes (e.g., "a string").

- Binding pattern annotations ($) must be used when appropriate. A $ in front of an attribute name indicates that the attribute has to be bound in that source.(i.e., it must have a constant value)

For the sources in Figure 11, we write the following source schema:

```
s1%s1(cr gc pn scc)
c1%c1(cc cn curr km lang)
s2%s2(pn)
s3%s3(gc)
s4%s4(cr gc pn)
s5%s5(cr gc pn tc)
s6%s6(cr pn)
s7%s7(dp pn)
c2%c2($km mi)
```

6.1.3 Axiom Definition

Each axiom has a left hand side and a right hand side. The former is expressed in domain terms, while the latter is expressed in source terms. The right hand side may contain a single source definition (*Simple Axiom*), a conjunction of source definitions (*Conjunctive Axiom*), or a disjunction of source definitions (*Disjunctive Axiom*).

```
<axiom> := <simple-axiom> | <conjunctive-axiom> | <disjunctive-axiom>
<simple-axiom> := <domain-concept>({<variable_name>}+) <->
                  <source-concept>({<variable_name>}+)

<conjunctive-axiom> := <simple-axiom> and
                       {<conjunctive-axiom> | <simple-axiom> | <constraint>}

<disjunctive-axiom> := { [<conjunctive-axiom>] | <simple-axiom> }
                       or
                       { [<conjunctive-axiom>] | <simple-axiom> }

<constraint> := <variable> <orderop> <constant>
<variable_name> := ?<string> | _ | ?$<string>
<constant> := <string> | <number>
<orderop> := = | < | > | <= | >=
```

- The number of variables for a domain concept must be identical to the number of domain attributes for that same concept as defined in the domain schema, so that the variable in position "i" maps to the domain attribute in position "i".
- The number of variables of a source concept must be identical to the number of source attributes for that same concept as defined in the source schema, so that the variable in position "i" maps to the source attribute in position "i".
- A $ in front of a variable name indicates that the corresponding attribute must be bound (i.e., it must have a constant value).
- A "_" in the place of a variable name indicates that this axiom does not provide information about the corresponding attribute.

- For <constraints>, if <constant> is numeric, any operator may be used. If <constant> is a string, only "=" is allowed.
- Binding pattern annotations ($) are used in the source clauses (right hand side) and in the domain clauses (left hand side) when the net result of applying all conjuncts would still require a particular attribute to be bound.

Example 1: Given the *domain schema*: `domain_concept(A B C)` and the *source schema*: `source1%source1($A B) source2%source2($B C)` the *axiom* should be:

```
domain(?$A ?B ?C) <->
   source1(?$A ?B) and
   source2(?$B ?C)
```

Note that ?A is marked as a binding variable since no source conjunct establishes it. You should not mark ?B since it is established by source1.

Example 2: If different orders of application can provide distinct axiom binding patterns, multiple axioms, one corresponding to each pattern, should in general be included. Given the *domain schema*: `domain_concept(A B C D)` and the *source schema*: `source1%source1($A B C) source2%source2(A $B D)` the *axioms* should be:

```
domain(?$A ?B ?C ?D) <->
   source1(?$A ?B ?C) and
   source2(?$B ?D)

domain(?A ?$B ?C ?D) <->
   source1(?$A ?B ?) and
   source2(?A ?$B ?D)
```

Simple Axiom Example:

```
seaport(_ ?cranes _ ?geoloc_code ?port_name ?seaport_country_code)
   <-> s1(?cranes ?geoloc_code ?port_name ?seaport_country_code)

small_seaport(_ ?cranes _ _ ?port_name _)
   <-> s6(?cranes ?port_name)

large_seaport(_ _ _ ?geoloc_code _ _)
   <-> s3(?geoloc_code)

country(?country_code ?country_name ?currency ?kilometers ?language _)
   <-> c1(?country_code ?country_name ?currency ?kilometers ?language)
```

Conjunctive Axiom Example:

```
small_seaport(_ ?cranes _ ?geoloc_code ?port_name ?seaport_country_code)
   <-> s1(?cranes ?geoloc_code ?port_name ?seaport_country_code)
       and s6(?cranes ?port_name)

small_seaport(_ ?cranes _ ?geoloc_code ?port_name ?seaport_country_code)
   <-> s1(?cranes ?geoloc_code ?port_name ?seaport_country_code)
       and ?cranes <= 7
```

```
large_seaport(_ ?cranes ?depth ?geoloc_code ?port_name ?seaport_country_code)
   <-> s1(?cranes ?geoloc_code ?port_name ?seaport_country_code)
       and s7(?depth ?port_name)

country(?country_code ?country_name ?currency ?kilometers ?language ?miles)
   <-> c1(?country_code ?country_name ?currency ?kilometers ?language)
       and c2(?kilometers ?miles)
```

Disjunctive Axiom Example:

```
seaport(_ _ _ _ ?port_name _)
   <-> s2(?port_name)
       or s6(_ ?port_name)

seaport(_ ?cranes _ _ ?port_name _)
   <-> s4(?cranes _ ?port_name)
       or s5(?cranes _ ?port_name _)
       or s6(?cranes ?port_name)

large_seaport(_ ?cranes ?depth ?geoloc_code ?port_name _)
   <-> [s4(?cranes ?geoloc_code ?port_name)
        and s7(?depth ?port_name)]
       or [s5(?cranes ?geoloc_code ?port_name _)
           and s7(?depth ?port_name)]
```

6.1.4 SQL Functions

7 The CSIMS Plan Language

We have seen in Section 1.1 that CSIMS can take as input a query, generate a plan for this query, and execute this plan to obtain the query results. In some cases, for instance if we have to execute a query over and over again, it might be desireable to give as an input to CSIMS the *plan* instead of the *query*. In this case, the plan generation code is skipped, and CSIMS just executes the given plan.

The CSIMS plan language is the following:

```
<CSIMS_Plan> := CSIMS_PLAN[ [<initial_node>] [<output_node>]
                {[<retrieve_node>] | {[<optional_node>]}+} ]

<optional_node> := <retrieve_node> | <join_node> | <select_node> |
                   <assignment_node> | <binary_union_node>

<initial_node> := <general_information>

<output_node> := <general_information>

<retrieve_node> := <general_information> Source:<string> SQL:<string>

<join_node> := <general_information> JoinConditions:{<join_expression>+}

<select_node> := <general_information> Selection:<select_expression>
```

```
<assignment_node> := <general_information> AssignmentExpression:<assignment_expression>

<binary_union_node> := <general_information>

<join_expression> := (<orderop> <variable> <variable>)

<select_expression> := (<orderop> <variable> {<variable>|<constant>})

<assignment_expression> := (:= <variable> {<string> | <arith_expression>})

<general_information> := ID:<int> TYPE:<node_type>
                         ProjectionVariables:{<variable>}+
                         From:<int>

<node_type> := InitialNode | output | retrieve |
               join | select | binary_union | assignment

<arith_expression> := <number> | <variable> |
                      (<arithop> <arith_expression> <arith_expression>)

<variable> := ?<string>
<constant> := <string> | <number>
<arithop> := + | - | * | /
<orderop> := = | < | > | <= | >=
```

- The ID of the *initial_node* is always 0.
- The ID of the *output_node* is always 1.
- *From* defines the edges in the plan. For example, "`From: 3`" denotes that there is an edge in the plan from the node with ID 3 to the current node.
- The *initial_node* always has an empty *From*.
- The *retrieve_nodes* always have "`From: 0`".
- The *ProjectionVariables* represent the variables that have to be present in the query result of a certain node.
- For a retrieve node we have to specify the SQL query and the source that provides information about this query.

For example, lets consider the query "list the geoloc codes, number of cranes and port name for all large seaports with more than 10 cranes", which has the following Loom representation:

```
(sims-retrieve(?gc ?cr ?pn)
  (:and (large_seaport ?ls)
        (geoloc_code ?ls ?gc)
(cranes ?ls ?cr)
(port_name ?ls ?pn)
(:= ?A 10)
(> ?cr ?A)))
```

and lets consider that the axiom used for this query is:

```
large_seaport(_ ?cranes ?depth ?geoloc_code ?port_name _)
   <-> [s4(?cranes ?geoloc_code ?port_name) and
        s7(?depth ?port_name)]
       or
       [s5(?cranes ?geoloc_code ?port_name _) and
        s7(?depth ?port_name)]
```

We can write the following plan according to the plan language specified above:

```
CSIMS_PLAN[
[
  ID : 0
  TYPE : InitialNode
  ProjectionVariable :
  From :
]

[
  ID : 1
  TYPE : output
  ProjectionVariables : ?geoloc_code0 ?cranes1 ?port_name2
  From : 2
]

[
  ID : 6
  TYPE : retrieve
  ProjectionVariables : ?csx_port_name3
  Source: s7
  SQL : "select distinct large_seaport2.pn from s7 large_seaport2"
  From : 0
]

[
  ID : 7
  TYPE : retrieve
  ProjectionVariables : ?port_name2 ?geoloc_code0 ?cranes1
  Source: s4
  SQL : "select distinct large_seaport1.pn, large_seaport1.gc,
           large_seaport1.cr from s4 large_seaport1"
  From : 0
]

[
  ID : 5
  TYPE : join
  ProjectionVariables : ?geoloc_code0 ?cranes1 ?port_name2
  JoinConditions : (= ?csx_port_name3 ?port_name2)
  From : 6 7
]
```

```
 [
   ID : 9
   TYPE : retrieve
   ProjectionVariables : ?csx_port_name4
   Source: s7
   SQL : "select distinct large_seaport4.pn from s7 large_seaport4"
   From : 0
 ]

 [
   ID : 10
   TYPE : retrieve
   ProjectionVariables : ?port_name2 ?geoloc_code0 ?cranes1
   Source: s5
   SQL : "select distinct large_seaport3.pn, large_seaport3.gc,
            large_seaport3.cr from s5 large_seaport3"
   From : 0
 ]

 [
   ID : 8
   TYPE : join
   ProjectionVariables : ?geoloc_code0 ?cranes1 ?port_name2
   JoinConditions : (= ?csx_port_name4 ?port_name2)
   From : 9 10
 ]

 [
   ID : 4
   TYPE : binary_union
   ProjectionVariables : ?geoloc_code0 ?cranes1 ?port_name2
   From : 5 8
 ]

 [
   ID : 3
   TYPE : assignment
   ProjectionVariables : ?geoloc_code0 ?cranes1 ?port_name2 ?a
   AxxignmentExpression : (:= ?a 10)
   From : 4
 ]

 [
   ID : 2
   TYPE : select
   ProjectionVariables : ?geoloc_code0 ?cranes1 ?port_name2
   Selection : (> ?cranes1 ?a)
   From : 3
 ]
 ]
```

8 Information-Source Wrappers

Once the CSIMS planner has selected the desired sources for a user's query and devised a plan for obtaining the required information, it must communicate with the individual information sources. Sometimes the information source may be complex and difficult to communicate with and additional data processing or functionality may be required. In order to modularize this process and cleanly separate query planning from communication issues, CSIMS requires that for each type of information source there exist a wrapper with which it will communicate. The purpose of the wrapper is to mediate between CSIMS and the information source. The wrapper must be capable of translating between the query language obtained from CSIMS and the information source's query language if necessary, as well as translating between the data output format of the information source and a format appropriate for CSIMS.

This section explains how wrappers are used by CSIMS. The first subsection describes the data that is communicated. The subsequent subsections describe the protocols by which the data is passed; KQML and CORBA.

8.1 Information Source Wrappers

An information source's wrapper will receive a query from CSIMS as input. The syntax of this query language can be varied, so long as the wrapper and CSIMS have agreed upon it. Predefined examples include the CSIMS query language or SQL. See section ?? for more on information sources. One restriction is that all concepts and roles used in the query will be drawn *only* from that information source. Note that at the time when such communication takes place CSIMS has already determined that the query being sent to the information source can be processed in its entirety by that source alone.

The information source's wrapper performs any necessary mediation between CSIMS and the information source. This may involve translating the query into the information source's particular query language, providing additional information to the information source or any other necessary reconciliation between CSIMS and the information source. It then submits a query to the source and retrieves the data. Next, it packages this data into a list of tuples corresponding to the variable parameters used in the submitted query. This tuple is then returned to CSIMS. See Figure 12.

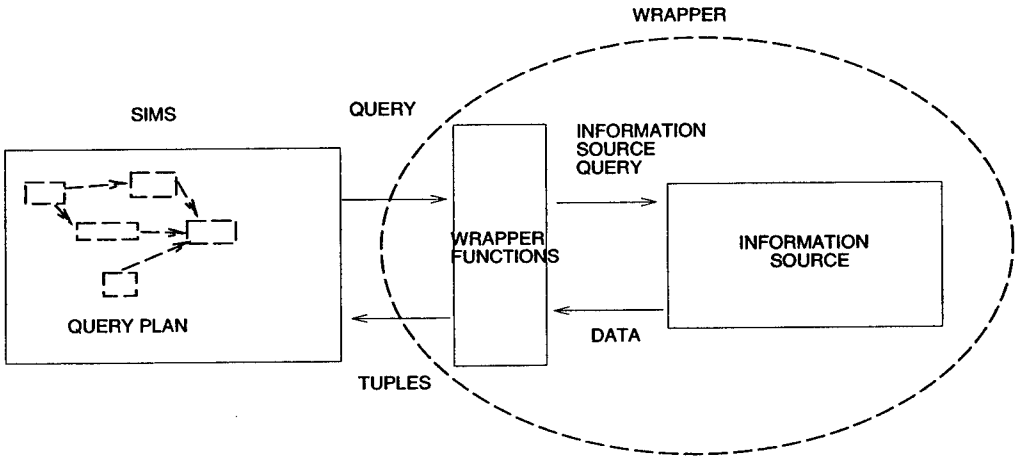


Figure 12: Data Flow between SIMS and Wrappers

In this way, CSIMS is insulated from the particulars of each information source. All the complexity of an information source is hidden from CSIMS via the wrapper module.

9 Communication Issues

9.1 Remote Communication Using KQML

Knowledge Query and Manipulation Language (KQML) protocol, is a language for communication and knowledge sharing between autonomous programs. A simplified view of KQML-based communication is presented in Figure 13.

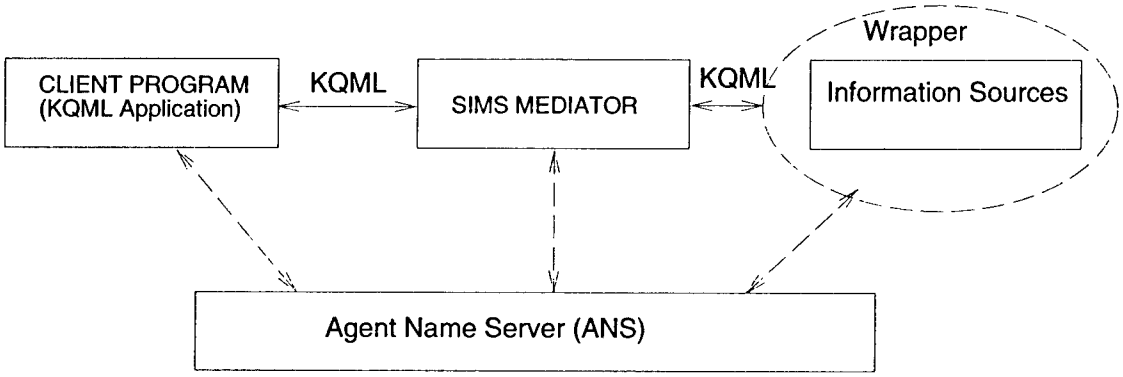


Figure 13: Communication via KQML

For our purpose, KQML provides two main types of functionality that ease the communication between clients and servers (KQML refers to both clients and servers as *agents*). KQML provides a flexible standard language for client-server communication that is available for many platforms, as well as implementation in different languages. It also provides a registry of all clients and servers, so that a client only need to refer to the name registered on the registry by the server (which is usually the name of the service provided and hence more meaningful than just a host address) to communicate with the servers.

The central registry of services in KQML is called the *agent name server (ANS)*, and it records all KQML agents and their addresses. We are mostly interested in the ANS for providing the addresses of information source wrappers CSIMS needs to communicate with (this address resolution process happens transparently and does not require user intervention). The client and server must both be registered with the ANS. The environment variable *KQML_ANS* specifies where the ANS is located, and both the client and server should agree on an ANS accessible to both. An agent is registered using a unique name of the following form, `<user>@<host>-<timestamp>`.

Note that the KQML clients/servers only contact the ANS once to verify the existence of a server and to get its address. The user need not know where a particular server is located but only its name (e.g., SQL-QUERY-SERVER). KQML transparently resolves the location through the ANS and caches each resolved location. The communication protocol used by KQML is TCP/IP. KQML creates a process that listens on a remote TCP/IP stream in order to detect messages from remote hosts.

The ANS used by KQML must be accessible to both CSIMS and to any users of the CSIMS system, but need not be run on those systems itself.

The client must know the messages supported by the server program because only those can be processed. In KQML terms, CSIMS acts as a mediator between the client and the information sources. A KQML mediator receives a request and either delegates it to one or more other servers, or processes it internally/locally (e.g., in a local database). Hence the information source server needs to define a handler for the messages it will support and the client needs to know these messages together with their form.

CSIMS currently uses the *:ASK-ALL* KQML performative to communicate with remote information source servers. The KQML message sent by CSIMS to the information source servers are of the form:

```
(:ASK-ALL :SENDER <SIMS server> :RECEIVER <info-source server>
          :REPLY-WITH T :CONTENT (<SQL query><hostname:dbname><username>))
```

9.2 Remote Communication Using CORBA

As CORBA [1] is supported by virtually all the industry leaders, making CSIMS a CORBA-compliant application broadens the area of potential CSIMS application. For instance, any CORBA-compliant application is able to act like a CSIMS client (i.e., to send queries to CSIMS and to receive the returned answers). A simplified view of CORBA-based communication is presented in Figure 14.

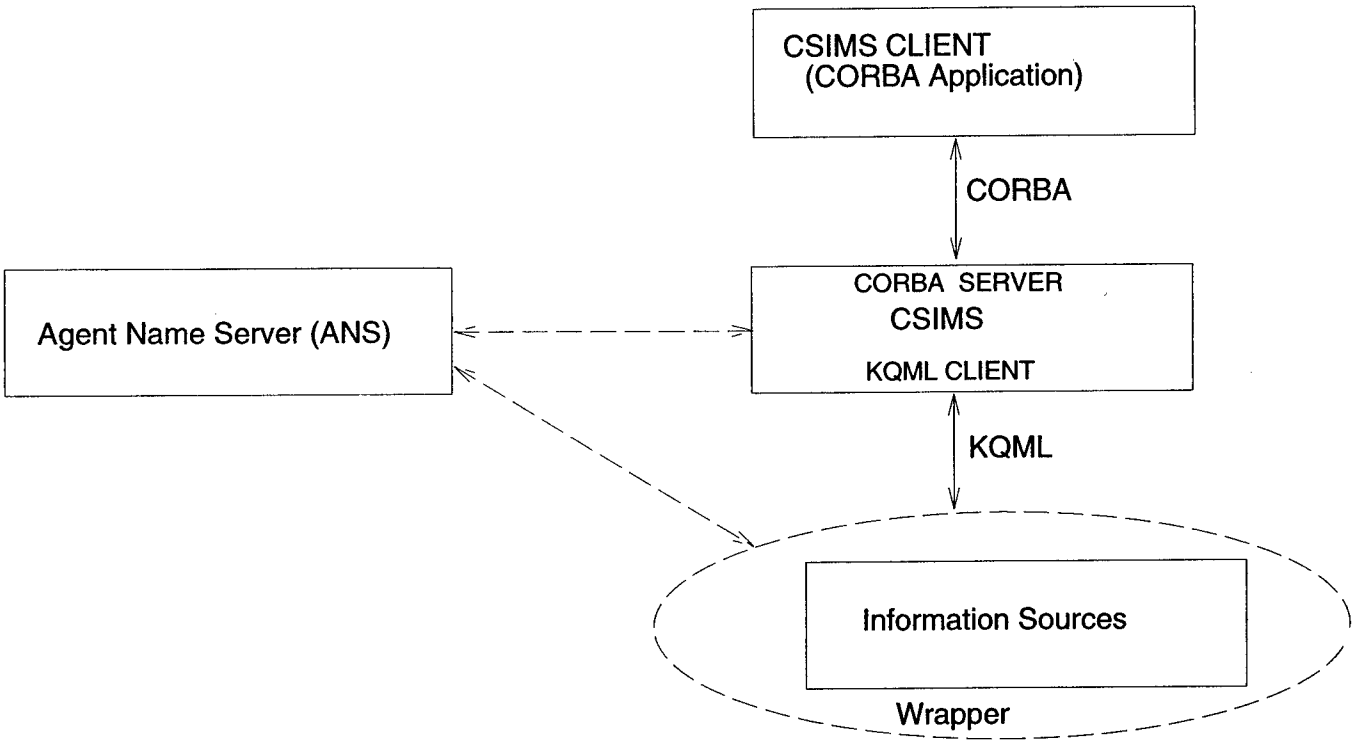


Figure 14: Communication via CORBA

CORBA defines distributed services for inter-process and inter-platform messaging, and it provides interoperability between applications written in different programming languages, running on different machines in heterogeneous, distributed environments. CORBA is an interoperability standard that has several implementations (e.g., Orbix, ILU, VisiBroker), and we currently support the Orbix 2.3 [2] implementation.

Based on our CSIMS-CORBA-Server, any number of CORBA-compliant applications can use CSIMS as a query-answering system.

Figure 15 shows a simple example of a CORBA client that reads a query specified as a command line argument and sends it to the CSIMS server named *CORBA2SIMS* which is located on the machine *vigor.isi.edu*.

```
main(int argc, char **argv)
{
  CORBA2SIMS* anObj = NULL;
  char *query;

  if (argc>=2)  {
    query=argv[1];
  } else {
    printf("Usage: client querystring");
    exit(1);
  }

  TRY{
      anObj = CORBA2SIMS::_bind(":CORBA2SIMS","vigor.isi.edu",IT_X);
  }
  CATCHANY{
      cout << "error" << IT_X << endl;
  }
  ENDTRY

  TRY{
      char * answer=NULL;

      long res = anObj->SendQuery(query, answer);
      if(answer!=0){
cout << answer;
CORBA::string_free(answer);
      }

      if(res!=1){
cout << "SendQuery returned error" << flush;
      }
  }
  CATCH(CORBA::SystemException &se){
      cout << "QQQQ #### CORBA_2_SIMS exception raised!!!" << endl;
      cout << endl << &se <<endl;
  }

}
```

Figure 15: CORBA client

10 Compiling and Running CSIMS

10.1 Compiling CSIMS

In order to compile CSIMS, make the required changes to the `Makefile` (read the file `COMPILE.info` that comes with the distribution), and execute one of the following commands:

- `make planner` : creates the executable `planner`, which is the stand-alone CSIMS;
- `make kqml` : creates the executable `server`, which is the version of CSIMS that can be accessed as a KQML agent;
- `make corba` : creates the executables `CORBA2SIMS` and `corba_client`. The former is the CSIMS CORBA server, while the latter is an example of a CORBA client application.

10.2 Configuring CSIMS

10.2.1 Required Files

CSIMS provides a configuration file that includes all CSIMS configuration variables. Before running CSIMS you must ensure that the system has access to an axiom file (AXIOM_FILE) that contains axioms for a specific domain (in the format specified in Section 6) and a information source file (IS_FILE) that contains the source declarations (see Section 4).

10.2.2 Output Formats

CSIMS can produce the resulted tuples in different formats. The variable OUTPUT_FORMAT specifies the output format. Available formats are: tab delimited format, generic OEM format and dynamic OEM format (the default is dynamic OEM format).

10.2.3 Optimization

By default, CSIMS generates an initial plan for your query, and executes that plan (replanning on failure). If you would like to optimize your initial plan using *Planning by Rewriting (PbR)*, you must set the variable REWRITE, and the variable NR_OF_NEW_PLANS. The higher the number of NR_OF_NEW_PLANS, the better the optimization.

10.3 Using KQML

If KQML is used in the system, set accordingly the KQML_HOME and KQML_HOST variables in the configuration file. KQML_HOST represents the hostname of the KQML ANS, (for details see Section 8).

To start an ANS execute the command:

```
($KQML_HOME)/bin/startans "hostname"
```

In order to check which agents are available in an ANS, or to verify that a service that was registered is up, run the following command in a UNIX shell:

```
($KQML_HOME)/bin/agentls
```

10.4 Using CORBA

If CORBA is used in the system, set accordingly the ORBIX_HOME variable in the configuration file.
To start the CORBA daemon, execute:

```
($ORBIX_HOME)/bin/orbixd
```

In order to check what servers are registered with the daemon, run the command:

```
($ORBIX_HOME)/bin/lsit
```

10.5 Trouble Shooting

In order to be able to handle errors more efficiently, we have created an error hierarchy that describes the types of errors currently handled by CSIMS. These errors are trapped throughout the execution process (by CSIMS itself, by the wrappers, etc) and sent back to the CSIMS server where they are handled appropriately. The current error hierarchy is described in Figure 16.

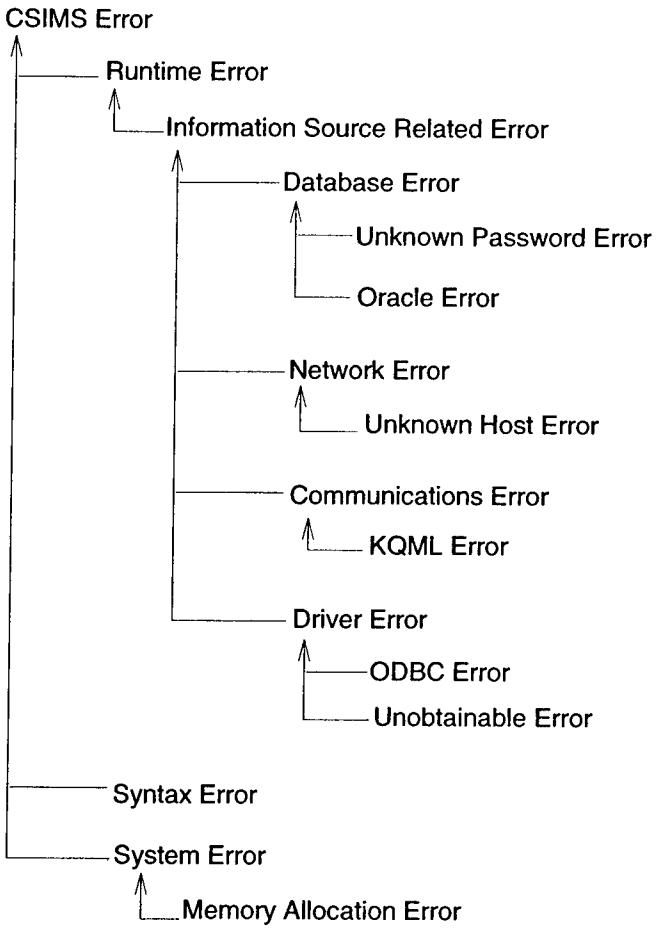


Figure 16: CSIMS Error Hierarchy

In case of an error, CSIMS will display an error message and exit. The error message should provide enough information for you to be able to eliminate the error. Common errors are *Syntax Error* and *Information Source Related Error. Syntax Error* occurs if the format of an axiom or the query is incorrect while *Information Source Related Error* can occur during the communicating with the information sources (for example, KQML is down, a database is down, a userid is incorrect, etc.).

CSIMS also provides a tracing mechanism. If the variable CSIMS_TRACE is set (i.e., values 1-7), tracing information is displayed. The higher the value of CSIMS_TRACE, the more information is displayed. The information can be shown on your screen or in a file (the default is on the screen). If the variable TRACE_FILE is set, all trace information is printed to that file.

10.6 Running CSIMS Standalone

To run CSIMS standalone, make the required changes to the configuration file (see Sections 10.2, 10.3, 10.4), execute the configuration file, and then execute the command:

```
planner "query_file"
```

where *planner* is the CSIMS executable, and *query_file* is a file that contains a query in LOOM or SQL syntax.

10.7 Running CSIMS as a KQML Agent

To run CSIMS as a KQML agent, make the required changes to the configuration file (see Sections 10.2, 10.3, 10.4), execute the configuration file, and then execute the command:

```
server "agent_name"
```

where *server* is the KQML agent executable, and *agent_name* is the name by which this agent will be registered with the ANS. The agent expects requests in the form:

```
(:ASK-ALL :RECEIVER <CSIMS agent>
          :REPLY-WITH T :CONTENT <query>)
```

where "CSIMS agent" is the "agent_name" and "query" is a SQL query.

10.8 Running CSIMS as a CORBA Server

To run CSIMS as a CORBA server, make the required changes to the configuration file (see Sections 10.2, 10.3, 10.4), execute the configuration file, and then register the CSIMS server with the Orbix daemon:

```
putit CORBA2SIMS ($CSIMS_HOME)/CORBA2SIMS
chmodit i+all
chmodit l+all
```

where *CORBA2SIMS* is the CSIMS server executable. Use the generated CORBA client program to test your server. Execute:

```
client "query"
```

where "query" is a SQL or LOOM query.

10.9 Running CSIMS from a CGI Script

CSIMS can also be accessed through HTTP requests that contain the query that CSIMS must execute. The CGI script invokes CSIMS as a stand-alone process (see Section 10.6) and handles the query result received from CSIMS. The CGI script can configure CSIMS according to Section 10.2 by setting the relevant environment variables from the configuration file.

10.10 Running CSIMS from the GUI

First, start the CSIMS server on some port number.

```
planner -port[port_number]
```

You have to have JDK1.2 installed in order to use GUI. Launching a GUI application is as simple as running simsgui.csh. If $JAVA_HOME environment variable is not set, the sript will ask you to provide it.

```
simsgui.csh
```

Applet version of CSIMS GUI is also available. Use the following template HTML tags to embed a CSIMS GUI applet in an HTML page. Because applet is not allowed to create socket connections to anywhere but the same host from where itself is downloaded, CSIMS server must run on the same host with the HTTP server.

```
<APPLET CODE=SIMS WIDTH=[ number in pixel ] HEIGHT=[ number in pixel ]>
<PARAM NAME=host VALUE="[ host name ]">
<PARAM NAME=port VALUE=[ port number ]>
</APPLET>
```

If your browser supports JDK1.2, you can view the HTML page in the browser, otherwise, use appletviewer that comes with JDK1.2. 'simsguiapplet.csh' is also included in the package.

```
simsguiapplet.csh
```

Users can either type the query (SQL/SIMS) directly to the query fields or select from the avaiable query list in the menu. GUI shows the graph representation of the query plan and also the execution status of the plan. The status of the information sources can also be manipulated through GUI. For detailed instruction on how to use, see `http://ariadne.isi.edu/simsgui.html`.

11 System Requirements

The CSIMS system currently runs in C++ under Solaris on SUN workstations. CSIMS requires the following software components:

KQML (Knowledge Query and Manipulation Language) provides remote communication support between CSIMS and remote DB servers, and between other information integration agents and CSIMS. We are currently using KQML version 2.06. Included with the CSIMS release are several patches which improve the performance of KQML in the CSIMS world. KQML is available by signing a license with University of Maryland, Baltimore County. For more information, see `http://www.cs.umbc.edu/kqml/`.

ODBC. We include with CSIMS code that provides a programmatic interface to Oracle databases using ODBC. We currently use ODBC 3.02 from InterSolv. This component is optional since CSIMS can be run with any database wrapper you may choose to implement.

CORBA. We also include with CSIMS code that allows you to communicate with CSIMS via CORBA. This is optional as well. It does require ORBIX version 2.3 from Iona Technologies to run. For more information on ORBIX, see `http://www.iona.com/`.

STL. CSIMS requires ObjectSpace's `C++ Standard<Toolkit>` version 2.1, which can be obtained from `http://www.objectspace.com/`.

FLEX++ and BISON++. CSIMS requires flex++ version 2.3.8-7 and bison++ version 1.2.1-8 for tokenization and parsing.

12 Coded Example

This section gives the code that implements the example discussed throughout the manual.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; domain-model.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Domain model for the example in the SIMS user manual
;;;

(in-package :sims)
(in-context :example)

;;; Seaport relations

(def-sims-relation geoloc-code
    :domain seaport
    :range string)

(def-sims-relation port-name
    :domain seaport
    :range string)

(def-sims-relation cranes
    :domain seaport
    :range number)

(def-sims-relation depth
    :domain seaport
    :range number)

(def-sims-relation tariff-code
    :domain european-large-seaport
    :range string)

(def-sims-relation seaport-country-code
    :domain seaport
    :range string)

;;; Seaport concepts

(def-sims-concept american-large-seaport
    :is-primitive large-seaport
    :annotations ((key (geoloc-code))
                  (key (port-name))))

(def-sims-concept european-large-seaport
    :is-primitive (:and large-seaport
                        (:the tariff-code string))
    :annotations ((key (geoloc-code))
                  (key (port-name))))

(def-sims-concept large-seaport
    :is (:and seaport
              (> cranes 7))
    :annotations ((key (geoloc-code))
                  (key (port-name))
                  (covering (american-large-seaport
                             european-large-seaport))))

(def-sims-concept small-seaport
    :is (:and seaport
              (<= cranes 7))
    :annotations ((key (geoloc-code))
                  (key (port-name))))

(def-sims-concept seaport
```

```
    :is-primitive (:and sims-domain-concept
                        (:the country-of country)
                        (:the geoloc-code string)
                        (:the seaport-country-code string)
                        (:the port-name string)
                        (:the cranes number)
                        (:the depth number))
    :annotations ((key (geoloc-code))
                  (key (port-name))
                  (covering (large-seaport small-seaport))))

;;; Country relations

(def-sims-relation country-code
    :domain country
    :range string)

(def-sims-relation country-name
    :domain country
    :range string)

(def-sims-relation currency
    :domain country
    :range number)

(def-sims-relation language
    :domain country
    :range number)

;;; Country concepts

(def-sims-concept country
    :is-primitive (:and sims-domain-concept
                        (:the country-code string)
                        (:the country-name string)
                        (:the language string)
                        (:the currency string))
    :annotations ((key (country-code))))

(def-sims-relation country-of
    :domain seaport
    :range country
    :is (:satisfies (?s ?c)
          (:for-some (?country-code)
            (:and (seaport ?s)
                  (country ?c)
                  (seaport-country-code ?s ?country-code)
                  (country-code ?c ?country-code)))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; queries.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Example queries
;;;

(in-package :sims)
(in-context :example)

(setq *queries*
  '(;; Large Seaport queries

    (11 "What the geoloc codes of all large seaports"
     (sims-retrieve (?geoloc-code)
       (:and (large-seaport ?ls)
        (geoloc-code ?ls ?geoloc-code))))

    (12 "How many cranes are available in various large seaports"
     (sims-retrieve (?cr)
       (:and (large-seaport ?ls)
        (cranes ?ls ?cr))))

    (13 "List the geoloc-codes and number of cranes for all large
seaports"
     (sims-retrieve (?geoloc-code ?cr)
       (:and (large-seaport ?ls)
        (geoloc-code ?ls ?geoloc-code)
        (cranes ?ls ?cr))))

    ...

    (106 "List currency and language for all countries"
     (sims-retrieve (?cc ?cn ?currency ?lang)
       (:and (country ?c)
        (country-code ?c ?cc)
        (country-name ?c ?cn)
        (currency ?c ?currency)
        (language ?c ?lang))))

    (107 "List all countries' currency, language, and seaport information"
     (sims-retrieve (?cc ?cn ?currency ?lang ?cr ?geoloc-code ?port-name)
       (:and (country ?c)
        (country-code ?c ?cc)
        (country-name ?c ?cn)
        (currency ?c ?currency)
        (language ?c ?lang)
        (seaport ?s)
        (cranes ?s ?cr)
        (geoloc-code ?s ?geoloc-code)
        (port-name ?s ?port-name)
        (country-of ?s ?c))))

    ))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; exkb1.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Example source model of EXKB1 for SIMS manual
;;;

(in-package :sims)
(in-context :example)

;;; Define a new Loom KB data source called EXKB1

(define-source EXKB1 loom-kb-source)

;;; Describe the domain in terms of EXKB1

(source-description seaport s1 EXKB1
  (geoloc-code gc)
  (port-name pn)
  (cranes cr)
  (seaport-country-code scc))

(source-description country c1 EXKB1
  (country-code cc)
  (country-name cn)
  (language lang)
  (currency curr))

;;; Load data (facts) into this new data source

;;; Seaport data

(deffact EXKB1 S1 ABIDJAN
  (PN "Abidjan")
  (GC "AAPV")
  (CR 5)
  (SCC "IV"))
...
(deffact EXKB1 S1 VLORE
  (PN "Vlore")
  (GC "YALP")
  (CR 1)
  (SCC "AL"))

;;; Country data

(deffact EXKB1 C1 ALBANIA
  (CC "AL")
  (CN "ALBANIA")
  (LANG "ALBANIAN")
  (CURR "LEK"))
...
(deffact EXKB1 C1 ZIMBABWE
  (CC "ZI")
  (CN "ZIMBABWE")
  (LANG "ENGLISH")
  (CURR "DOLLAR"))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; exdb.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Example source model of EXDB for SIMS manual
;;;

(in-package :sims)
(in-context :example)

;;; Define a new SQL database, addressed using ODBC alias name,
;;; communication via KQML, called EXDB

(define-source EXDB kqml-odbc-sql-source
  :host "isd18.isi.edu"
  :agent-name "sql_server"
  :db-name "examplei"
  :userid "ifd")

;;; Describe the domain in terms of EXDB

(source-description large-seaport lgsp exdb
  (geoloc-code gc)
  (depth dp)
  (port-name pn)
  (cranes cr)
  (seaport-country-code scc))

(source-description small-seaport smsp exdb
  (geoloc-code gc)
  (port-name pn)
  (cranes cr)
  (seaport-country-code scc))

(source-description european-large-seaport lgeurosp exdb
  (geoloc-code gc)
  (depth dp)
  (port-name pn)
  (cranes cr)
  (tariff-code tc)
  (seaport-country-code scc))

(source-description american-large-seaport lgamersp exdb
  (geoloc-code gc)
  (depth dp)
  (port-name pn)
  (cranes cr)
  (seaport-country-code scc))

(source-description seaport sp exdb
  (geoloc-code gc)
  (port-name pn)
  (cranes cr)
  (seaport-country-code scc))

(source-description country ctry exdb
  (country-code cc)
  (country-name cn)
  (language lang)
  (currency curr))

;;; No KB facts
```

13 Additional Reading

Using this manual and following the instructions in it require familiarity with SIMS, as well as with the Loom knowledge representation language, and the KQML transport protocol.

The following papers may be consulted for further information about these programs.

13.1 SIMS

1. Ambite, J.L. and Craig A. Knoblock Reconciling Distributed Information Sources. *Working Notes of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments*, Palo Alto, CA, 1995.

2. Ambite, J.L., Yigal Arens, Naveen Ashish, Chin Y. Chee, Chun-Nan Hsu, Craig A. Knoblock Wei-Min Shen, and Sheila Tejada. 1995. *The SIMS Manual*, Version 1.0. ISI/TM-95-428.

3. Arens, Y., Craig A. Knoblock and Chun-Nan Hsu. Query Processing in the SIMS Information Mediator. *Advanced Planning Technology*, editor, Austin Tate, AAAI Press, Menlo Park, CA, 1996.

4. Arens, Y., Chee, C.Y., Hsu, C-N., and Knoblock, C.A. 1993. Retrieving and Integrating Data from Multiple Information Sources. In *International Journal of Intelligent and Cooperative Information Systems*. Vol. 2, No. 2. Pp. 127-158.

5. Arens, Y., Knoblock, C.A., and Shen W-M. Query Reformulation for Dynamic Information Integration, *Journal of Intelligent Information Systems*, 6(2/3):99-130, 1996.

6. Arens, Y. and Knoblock, C.A. 1994. Intelligent Caching: Selecting, Representing, and Reusing Data in an Information Server. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM-94)*, Gaithersburg, MD.

7. Arens, Y., Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock Retrieving and Integrating Data from Multiple Information Sources. International Journal of Intelligent and Cooperative Information Systems. Vol. 2, No. 2. Pp. 127-158, 1993.

8. Arens, Y. and Knoblock, C.A. 1992. Planning and Reformulating Queries for Semantically-Modeled Multidatabase Systems, *Proceedings of the First International Conference on Information and Knowledge Management (CIKM-92)*, Baltimore, MD.

9. Hsu, C-N., and Knoblock, C.A. 1995. Estimating the Robustness of Discovered Knowledge, in *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, Montreal, Quebec, Canada.

10. Hsu, C-N., and Knoblock, C.A. 1995. Using inductive learning to gen- erate rules for semantic query optimization. In Gregory Piatetsky-Shapiro and Usama Fayyad, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 17. MIT Press.

11. Hsu, C-N., and Knoblock, C.A. 1994. Rule Induction for Semantic Query Optimization, in *Proceedings of the Eleventh International Conference on Machine Learning (ML-95)*, New Brunswick, NJ.

12. Hsu, C-N., and Knoblock, C.A. 1993. Reformulating Query Plans For Multidatabase Systems. In *Proceedings of the Second International Conference of Information and Knowledge Management (CIKM-93)*, Washington, D.C.

13. Hsu, C.-N. and Knoblock, C. A. Discovering Robust Knowledge from Dynamic Closed-World Data. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, 1996.

14. Knoblock, C.A., Arens, Y. and Hsu, C-N. 1994. An Architecture for Information Retrieval Agents. In *Proceedings of the Second International Conference on Cooperative Information Systems*, University of Toronto Publications, Toronto, Ontario, Canada.

15. Knoblock, C.A. 1995. Planning, Executing, Sensing, and Replanning for Information Gathering. In *IJCAI-95*, Montreal, Quebec, Canada.

16. Craig A. Knoblock Applying a General-Purpose Planner to the Problem of Query Access Planning. *Proceedings of the AAAI Fall Symposium on Planning and Learning: On to Real Applications*, 1994.

17. Knoblock, C.A. 1994. Generating Parallel Execution Plans with a Partial-Order Planner. *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference (AIPS94)*, Chicago, IL.

18. Craig A. Knoblock Building a Planner for Information Gathering: A Report from the Trenches Artificial Intelligence Planning Systems: *Proceedings of the Third International Conference* (AIPS96), Edinburgh, Scotland, 1996.

19. Craig A. Knoblock and Jose Luis Ambite. Agents for Information Gathering *Software Agents*, J. Bradshaw ed., AAAI/MIT Press, Menlo Park, CA, 1997.

20. Craig A. Knoblock, Yigal Arens, and Chun-Nan Hsu. Cooperating Agents for Information Retrieval. *Proceedings of the Second International Conference on Cooperative Information Systems*, Toronto, Ontario, Canada, University of Toronto Press, 1994.

These publications, as well as additional information about SIMS, can be accessed through the WWW at `http://www.isi.edu/sims/`.

13.2 Loom

1. MacGregor, R. A Deductive Pattern Matcher. In *Proceedings of AAAI-88, The National Conference on Artificial Intelligence*. St. Paul, MN, August 1988.

2. MacGregor, R. The Evolving Technology of Classification-Based Knowledge Representation Systems. In John Sowa (ed.), *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann. 1990.

Additional papers and information about Loom can be accessed trough the WWW at the Loom Project homepage: `http://www.isi.edu/isd/LOOM/LOOM-HOME.html` .

13.3 KQML

1. Finin, T., Fritzson, R. and McKay, D. A Language and Protocol to Support Intelligent Agent Interoperability. In *Proceedings of the CE and CALS Washington '92 Conference*, June, 1992.

Additional papers and information about KQML can be accessed through the WWW at the KQML homepage: `http://www.cs.umbc.edu/kqml/` .

13.4 CORBA related

1. Iona Technologies. Orbix 2.2: Programming Guide. March 1997.

2. Object Management group. The Common Object Request Broker: architecture and specification. OMG Document Number 91.12.1, 1991.

Acknowledgements

We would like to thank the developers of the software systems that we have used extensively in the construction of SIMS. In particular, thanks to Bob MacGregor and Tom Russ for the Loom knowledge representation system. Thanks to Don McKay, Jon Pastor, and Robin McEntire at Paramax/Unisys/Loral for their implementation of the KQML language. And thanks to Dan Weld and Tony Barrett at the University of Washington for the UCPOP planner, which we used to build the SIMS planner. In addition, thanks to Ping Luo for his testing of and feedback on an earlier version of this manual.

References

[1] Object Management group. *The Common Object Request Broker: architecture and specification*, volume OMG Document Number 91.12.1. Object Management group, 1991.

[2] Iona Technologies. *Orbix 2.2: Programming Guide*. Iona Technologies, March 1997.

APPENDIX II: WARFIGHTER S INFORMATION PACKAGER

Warfighter s Information Packager

Appeared in:

Proceedings of *the Tenth Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-98).*

Madison, Wisconsin, July 27-29, 1998.

Pp. 1095—1100.

Warfighter's Information Packager

Yigal Arens
Weixiong Zhang
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
{arens,zhang}@isi.edu

Yongwon Lee
Jon Dukes-Schlossberg
Lockheed Martin
Intelligent Systems Center
3251 Hanover St. H1-43
Palo Alto, CA 94304
{ylee,slosh}@ict.atc.lmco.com

Marc Zev
ISX Corporation
4353 Park Terrace Dr.
Westlake Village, CA 91361
mzev@isx.com

Abstract

The Warfighter's Information Packager (WIP) is a suite of distributed components that allows users to easily obtain information from diverse heterogeneous data sources and to display the results in a user-defined predictable manner.

WIP is based on research performed under DARPA's Intelligent Integration of Information program. WIP uses a combination of AI and non-AI technologies to take advantage of the information push technology being developed for DARPA's Battlefield Awareness and Data Dissemination (BADD) program, and being deployed during the Fall of 1998.

Together, the WIP components create a distributed system that serves as a valuable tool for information analysis by: 1) allowing the user to define high-level information products, *information packages,* which are parameterized by user interests and specific tasks and roles; 2) providing a web-based package viewer that dynamically constructs packages for the user on demand and performs value-added information *linking*; 3) allowing users to make high-value complex information requests that can span multiple data sources without a priori knowledge of the schema of the sources,; and 4) monitoring data sources and *anticipating* useful modifications to a user's information package.

1 Problem Description

The Warfighter's Information Packager (WIP) system addresses the problem of how to support the needs of users to view and manipulate required data in an information push environment. In such an environment, traditional query-based data retrieval is replaced by asynchronous information delivery based on information profiles registered with sources.

Information push is intended to address the following problem: When information is generated in many sources and on a continual basis, it is onerous and inefficient to require users to issue multiple repeated queries for their information. It would be far more efficient if responsibility for disseminating the information were transferred to the sources. The sources contain the actual information and can more readily detect changes. In such an architecture, the consumer registers an *information profile* with the source, or with a system overseeing the source, that describes the kind of information they need.

There are many ways in which such a scheme may be implemented. The BADD program was established to deal with the issue of information push in a battlefield environment. BADD has chosen an architecture wherein profiles are registered with a central Information Dissemination Manager (IDM), and the IDM ensures that data matching the profiles is transferred to the deployed sources. This still leaves open the questions of how to construct and use applications that require the pushed data, and how to determine which profiles must be registered in order to support these applications. The WIP system addresses these issues.

WIP provides end-users with a facility for producing an *information package.* WIP, using IDM, ensures that necessary data is pushed to deployed sources when it is available. An *Information Integrator* satisfies information package queries by retrieving data from the deployed sources. Throughout both the specification and utilization of the product, an *Anticipator* component observes user and world events and may trigger the modification of existing packages. This capability allows profiles to be updated dynamically, in response to changing circumstances.

The specific problems that WIP addresses are:

1. The warfighter has a task to perform and has specific information needs. Information needs in BADD are specified as profiles that do not necessarily match up conveniently with actual source queries. This is due to the profile registration facility of the IDM having a language far less powerful than familiar database query languages. Furthermore, the context in which the task ends up being performed may require variation of the profile—something that is hard for the end-user to anticipate.

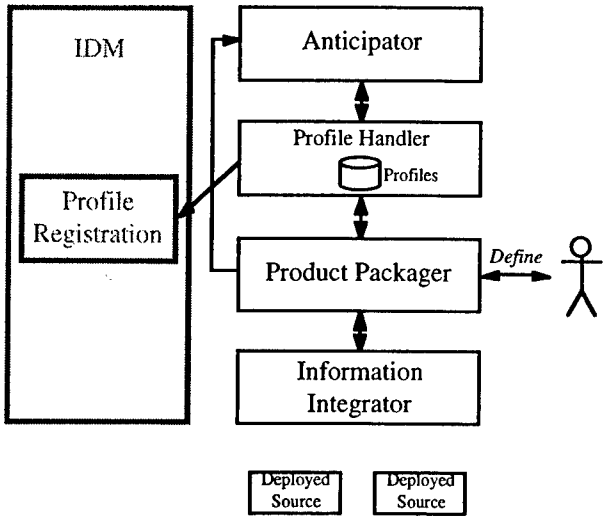


Figure 1: Information Package Specification

2. The data is delivered to the end-user's computer environment in formats that may differ from one environment to the next due to variations in data management facilities. For example, one warfighter may have a database facility available for a certain type of data, while another may not. In such a situation the IDM will deposit the information in a file. For similar reasons, data storage formats used in the deployed sources may differ from those used in original sources against which the profiles are registered. In addition, data needed for a single task may end up being distributed over different deployed sources. The end result is that any processing of deployed information will necessarily involve an information integration and/or translation task. Making WIP packages portable requires that this integration be done in a general manner and that queries to the information integrator be at a semantic level, independent of precise sources used and their organization.

We have chosen to use selected AI technologies, some developed as part of DARPA's Intelligent Integration of Information (I3) program, to address this problem.

2 Application Description

This section describes WIP's *product specification phase* and *product utilization phase*. Both phases complement each other, and the IDM, to provide an end-to-end information push system.

2.1 Information Package Specification

Figure 1 presents the architecture of the product specification phase. During this phase, the user interacts with the Product Packager to define the information package and the information profiles ultimately required for its successful utilization. It consults the Information Integrator in order to determine which sources are to be monitored. Since the local, deployed sources are mirrors of the original sources accessible through the IDM, (even if their organization and DBMS may differ) the Information Integrator can provide the Product Packager with necessary information to locate the appropriate sources. Defined profiles are handed to the Package Handler for local storage and for IDM registration. They are also handed to the Anticipator to infer user and world events that would necessitate modification of the information package.

The following sections describe the operations and capabilities of each of the modules in Figure 1 in more detail.

2.1.1 Product Packager

The Product Packager: 1) allows the user to define high-level information products, *information packages*, which are parameterized by user information needs and specific tasks and roles; 2) describes how the information should be formatted when presented to the user; 3) provides a web-based viewer that dynamically constructs packages for the user on demand and performs value-added information linking.

The Product Packager module, developed at ISX, is composed of two main components: the Package Editor and the Product Viewer. The Package Editor provides an environment for creating information package templates. These templates include parameterized queries and display formatting information. WIP's query description methodology allows users to easily describe their information needs based on a descriptive domain ontology. WIP's parameterization allows the dynamic modification of the precise semantics of queries without revising queries. This is accomplished by providing late-time binding of variables within the query expressions, enabling information requirements such as: "Get the target information for all targets on today's target list" to be expressed.

The Package Editor also performs the registration of necessary data profiles so that data can be deployed to the user's system. The Product Viewer, discussed more thoroughly in Section 2.2.1, collects the information results from the information integrator module, formats them based on the format defined in the package template, and displays the packaged product to the user.

2.1.2 Information Integrator

The Information Integrator module is an application of SIMS AI technology, developed at USC/ISI. SIMS serves as a single access point for information distributed over a collection of heterogeneous data sources. The underlying technology is described in Section 3.2. During the specification phase, the Information Integrator uses its knowledge of the distribution of data over sources accessible to the IDM (also mirrored, as noted earlier, by the distribution of potential data over local deployed

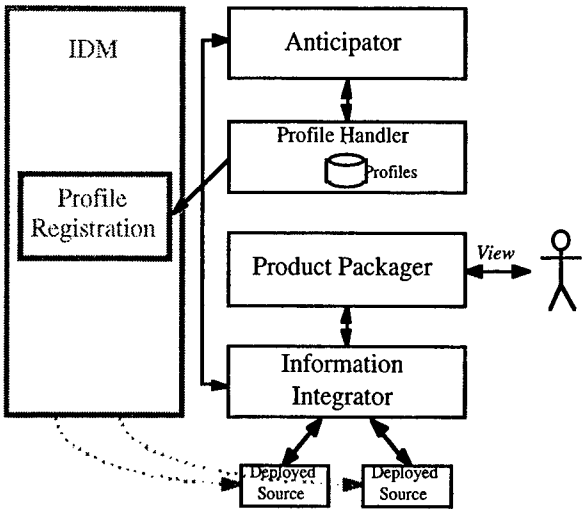


Figure 2: Information Package Utilization

sources) to provide the Product Packager with the identity of sources against which profiles must be registered. This enables the package being defined to have the information required for its operation at utilization time. Furthermore, SIMS' capability to determine which queries will have to be asked of each component source to satisfy the package's needs is used by the Product Packager. The package defines the profiles that will eventually be handed to the Package Handler for processing and registration.

2.1.3 Profile Anticipator

The Profile Anticipator dynamically updates user information needs in response to circumstances that have changed since the original information profile was defined. It handles conditional and unexpected information needs of users, once their basic information needs are registered. "Basic" information refers to required information to be delivered at all times, while "conditional" information refers to data to be delivered only when a certain condition is met (e.g., only when the mission is completed, send me the video image of the target). The Anticipator is an integral part of the WIP system not only to achieve dynamic intelligent information packaging but also to simplify the complexity of package entry.

Upon notification of an information profile registration, the Anticipator examines its contents and generates an anticipator profile for that user. An anticipator profile specifies the conditions under which the user's profile needs to be modified. For example, as a user moves from one location to another, the Anticipator monitors the user's current location to update information needs that may change. The anticipator profile is registered through the Product Packager just as the user's information profile is registered. The Anticipator then generates a package monitor that continuously and periodically requests the data for this package from the Product Packager to check the occurrence of "significant" and "interesting" events. When such events are detected, the monitor notifies the Anticipator that then carries out appropriate actions, i.e., the user's information profile is updated. When a user's information profile is deactivated, the associated anticipator profile and its package monitor are also deactivated.

The Anticipator maintains a rule base to provide domain-specific knowledge necessary for generating the anticipator profiles and to specify definitions of "interesting" events and associated appropriate actions. The Anticipator rules are represented in OEM (Object Exchange Model [Goldman et al. 96]), and implemented in Java.

2.1.4 Package Handler

The Package Handler maintains a local store of profiles defined specifically by WIP, handles communications, and registers and de-registers these profiles with the IDM.

2.2 Information Package Utilization

In the package utilization phase, packages are run and their information can be viewed and manipulated by the end-user. The package typically needs specific information—it requests data from the Information Integrator, which queries the deployed sources. The Profile Anticipator will monitor those same deployed sources via the Information Integrator in order to update users' information profiles, if and when necessary.

Figure 2 presents the architecture for the information package utilization phase. The following sections describe the operations and capabilities of each of the modules in more detail.

2.2.1 Product Packager

During the package specification phase, the Package Editor was responsible for user interactions. However, during the package utilization phase the Product Viewer is the focus of the system's operation. During the package specification phase the Product Packager enhances a user's ability to analyze information by allowing the creation of query expressions based on a high-level domain model, and by defining the formats of results both on a query by query basis and the package as a whole. In this phase, the Product Packager manages the returned information in order to provide the user with additional semantic insight to the data.

Using the Product Viewer, the user requests that a package be displayed. The Product Packager retrieves the requested package from the Package Handler and submits the package's queries to the Information Integrator. Upon receipt of the queried information, the Product Packager aids the user by identifying semantic links among the user's requested information and that requested by others, including information gathered on behalf of the Anticipator. The Product Packager then formats the information as specified in the package and renders the package, dynamically, based on the user's currently

available bandwidth and display options, using the Product Viewer.

While formatting and rendering the package, the Product Packager adds hyperlinks to the value-added linked information.

2.2.2 Information Integrator

In this phase the Information Integrator performs its "standard" function of query answering. It receives queries from the Product Packager and the Anticipator for data in the local sources, and provides it. The queries are in a high-level language using terms from the domain model (see Section 3.2 for details), hiding from the other WIP modules the details of source distribution, organization and language. The same package will work in different environments as long as the Information Integrator's description of the available sources is revised to reflect the specifics of the local situation.

2.2.3 Profile Anticipator

Because the Anticipator can add and update a user's information needs, information packages can contain new information the user does not specify in his/her profile. The operation of the Profile Anticipator is not modified during the package utilization stage, and it performed its monitoring of data sources regardless of user interactions with the Product Viewer.

2.2.4 Package Handler

During the utilization phase, the Package Handler acts the same as during the specification phase.

3 Use of AI Technology

3.1 Product Packager and Profile Editor

3.1.1 Information Package Template Selection

In military situations, such as the one WIP is designed for, it is typical that many users have the same or a similar data capacity. For this reason and for the efficiency of use, having pre-staged information package templates (IPTs) would be advantageous. However, because it is impossible to predetermine the information needs of every user, a better way is to establish IPTs that reflect the information needs based on an encapsulation of a particular user role. For instance, it would be far easier to determine the general information needs of someone in the Bosnian theater and store that as a specific IPT, store the information needs of a tank commander in a separate IPT, etc. Then, when the user needs to register an information package he needs only describe himself by his roles (e.g. tank commander in the Bosnian theater) and a specialized IPT can be constructed based on the aggregation of smaller, specific information package templates.

The ability to perform the aggregation of IPTs relies on the definition of roles for both the users and the IPTs. The purpose of a role is to provide a meaningful way of recognizing an IPT by the type of information requests contained within it. An IPT's role does not necessarily identify it uniquely, but it serves as a way to measure the commonality between that IPT and others. The idea is to define a role as some set of attributes common to all IPTs. Once a role is established for an IPT it is used as the index to find it again.

Collecting the appropriate IPTs for a user is not as simple as performing a *find* in a relational database. The determination of which IPTs are suitable can be based on a data-driven, rule-based methodology. Within military operations there are very specific rules about how information can flow, typically this means up through the chain of command. An example how this might work is as follows: a tank company commander in the Bosnian theater is interested in creating an appropriate IPT. An appropriate set of IPT might be: one for tank commanders, one for the Bosnian theater, one for the army. However, an IPT for a battalion would not be appropriate. The battalion commander in the same theater might get the tank commander's IPT plus IPTs for artillery companies. But if no IPT exists for the battalion level for Bosnia, a battalion level IPT for Europe might be appropriate. The IPT selection process is defined within an expert system, using a rule-based domain ontology.

3.1.2 Semantic Linking of Information

The Product Packager manages the information that is returned from the information integrator for all users of the WIP system at a given deployed site. As the Package Products are requested and viewed via the Product Viewer, a request is made to the Information Integrator to gather the data. Using a rule-based domain ontology as a guide, the Product Packager reviews all the data gathered from all users and makes a determination whether the data is relevant to the information requested in the original package. For instance, if the package has been registered to a tank unit and weather data has been collected and it is raining, the Product Packager would determine that road conditions would be a useful piece of information to include. If road conditions for the appropriate geographical area are available, then the Product Packager will add a hyper-inked connection to the original package pointing to the value added data.

3.2 Information Integrator

The Information Integrator module is an application of SIMS technology in the battlefield data dissemination domain. The overall goal of the SIMS project at USC/ISI is to provide integrated access to information distributed over multiple, heterogeneous sources: databases, knowledge bases, flat files, Web pages, programs, etc. In providing such access, SIMS insulates human users and application programs from the need to be aware of the location of sources and distribution of queried data over

them, individual source query languages, their organization, data model, size, and so forth. The processing of user requests should be robust, capable of recovery from execution-time failures and able to handle and/or report inconsistency and incompleteness of data sources. At the same time SIMS has the goal of making the process of incorporating new sources as simple and automated as possible.

The SIMS approach to this integration problem has been based largely on research in Artificial Intelligence, primarily in the areas of knowledge representation, planning, and machine learning. A model of the application domain is created, using a knowledge representation system to establish a fixed vocabulary for describing objects in the domain, their attributes and relationships among them. Using this vocabulary, a description is created for each information source. Each description indicates the data model used by the source, the query language, network location, size estimates, etc., and describes the contents of its fields in relation to the domain model. SIMS' descriptions of different information sources are independent of each other, greatly easing the process of extending the system. Some of the modeling is aided by source analysis software developed as part of the SIMS effort.

Queries to SIMS are written in a high-level language (a subset of SQL or Loom) using the terminology of the domain model – independent of the specifics of the information sources. Queries need not contain information indicating which sources are relevant to their execution or where they are located. Queries do not need to state how information present in different sources should be joined or otherwise combined or manipulated.

SIMS uses a planner to determine how to identify and combine the data necessary to process a query. In a pre-processing stage, all data sources possibly relevant to the query are identified. The planner then selects a set of sources that contain the queried information and generates an initial plan for the query. This plan is repeatedly refined and optimized until it meets given performance criteria. The plan itself includes, naturally, sub-queries to appropriate information sources, specification of locations for processing intermediate data, and parallel branches when appropriate. The SIMS system then executes the plan. The plan's execution is monitored and replanning is initiated if its performance meets with difficulties such as unexpectedly unavailable sources. It is also possible for the plan to include explicit replanning steps, after reaching a state where more is known about the circumstances of plan execution.

Changes to information sources are handled by changing source descriptions only. The changes will automatically be considered by the SIMS planner in producing future plans that utilize information from the modified sources. This greatly facilitates extensibility.

A variety of detailed publications describing SIMS is available (e.g., [Arens et al 96], [Arens et al 93], [Knoblock 95], [Knoblock 94]).

3.3 Profile Anticipator

The current prototype Anticipator module, developed at LM/ISC, implements data-driven, rule-based anticipation of information needs: data sources are continuously monitored for occurrences of interesting events. When such events occur, one or more anticipation rules are fired to update user's information needs. Researchers in both database (source subscription) and AI (knowledge engineering and representation) have enabled the current approach.

The anticipator uses domain-specific rules to determine data sources to be monitored, conditions (interesting events) to be checked, and actions to be executed when conditions are met. The anticipator rules are obtained through standard knowledge engineering techniques with military experts. All rules are deterministic, i.e., there is no probabilistic inference. The rules are described using high-level domain terms and the translation of these terms to source specific terms is performed by the Information Integrator module. A key feature of anticipator rules is their expressive event descriptions. Currently, a limited set of event descriptions is possible including comparisons of values of multiple attributes in two information packages.

The rules are represented in OEM, taking advantage of its flexibility (less structural constraints) and self-declarative and self-describing features. It is worth noting that anticipator rules contain neither data source nor implementation specific information. For example, to generate source monitors, rules specify sources and possibly attribute names, but not database queries in some query language (e.g., SQL).

The WIP anticipator module along with the Package Handler is an extension of Profile and Event Manager components of Intelligent Information Dissemination Server (IIDS) [Dukes-Schlossberg et al., 97], and is an implementation of an ongoing effort toward a fully automated information profiling system. We are currently designing an advanced information profiling system, which uses both Bayesian networks and anticipation rules to predict user's information needs from their identity information such as user type, location, status, and mission. That is, rather than explicitly asking what information is needed, users' information needs are predicted and their information profiles automatically constructed from what and where they are, and what mission they are engaged in.

4 Application Innovation

The design of the BADD system, as envisioned by DARPA and executed under its guidance, does not involve

any technology that could be classified as part of Artificial Intelligence. The participants in the effort described in this paper found that the incorporation of additional technology based on AI research and techniques could relatively easily (e.g., using an order of magnitude smaller funding) provide a very substantial added value. The information integration functionality provided by the Information Integrator, which uses AI planning technology, enables WIP to be operational on different information sources in multiple existences, including databases, legacy information systems and the World Wide Web. The semantic information linking function of Product Packager and anticipating function provided by the Anticipator, both of which use domain ontology and rule-based AI technologies, significantly extend WIP's usability to the end users. Combined, these two functions not only provide information that directly meets users' needs, but also supply and present context and other related information that makes the required information meaningful.

We would also like to point out that although WIP is developed for the BADD system in a military domain, the concept and WIP system itself can be easily applied to other applications. Specifically, WIP can be adapted to a personal digital assistant in an information rich environment to help a user collect information based on particular information needs. The information can then be presented in a format compliant with the user's preferences.

5 Evaluation

The WIP system is the integration of three distinct technologies: information integration, product packaging and information anticipation. The development of each of these technologies has been pursued independently, and the integration of components has taken place in the past between some of the contributors. Prototype systems with limited functionality exist for the components of the Product Packager. The Information Integrator (SIMS), responsible for satisfying information requests, is currently supporting several projects in extended prototype form. The Anticipator, which is responsible for anticipating users' information needs, is currently in development. The development schedule currently places the integration of WIP with the rest of the BADD system during the Summer of 1998. BADD will then be deployed as a working system in the Fall of 1998.

References

[Arens et al. 96] Yigal Arens, Craig A. Knoblock, and Wei-Min Shen. *Query Reformulation for Dynamic Information Integration.* Journal of Intelligent Information Systems, Vol. 6, No. 2/3, 1996.

[Arens et al. 93] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. *Retrieving and Integrating Data from Multiple Information Sources.* International Journal of Intelligent and Cooperative Information Systems. Vol. 2, No. 2. Pp. 127-158, 1993.

[Dukes-Schlossberg et al. 97] Jon Dukes-Schlossberg, Yongwon Lee, and Nancy Lehrer. *IIDS: Intelligent Information Dissemination Server.* Proceedings of MILCOM 1997.

[Goldman et al 96] Roy Goldman, Sudarshan Chawathe, Auturo Crespo, Jason McHugh. *A Standard Textual Interchange Format for the Object Exchange Model (OEM).* Technical Report, Stanford University, 1996.

[Knoblock 95] Craig A. Knoblock. *Planning, Executing, Sensing, and Replanning for Information Gathering.* Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Montreal, Canada, 1995.

[Knoblock 94] Craig A. Knoblock. *Applying a General-Purpose Planner to the Problem of Query Access Planning.* Proceedings of the AAAI Fall Symposium on Planning and Learning: On to Real Applications, 1994.

*U.S. GOVERNMENT PRINTING OFFICE: 2000-610-055-10034

DISTRIBUTION LIST

addresses	number of copies
DR. RAYMOND A. LIUZZI AFRL/IFTD 525 BROOKS ROAD ROME, NY 13441-4505	10
USC INFORMATION SCIENCES INSTITUTE 4676 ADMIRALTY WAY MARINA DEL RAY, CA 90292-6695	5
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	1
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
ATTN: NAN PFRIMMER IIT RESEARCH INSTITUTE 201 MILL ST. ROME, NY 13440	1
AFIT ACADEMIC LIBRARY AFIT/LDR, 2950 P.STREET AREA B, BLDG 642 WRIGHT-PATTERSON AFB OH 45433-7765	1
AFRL/MLME 2977 P STREET, STE 6 WRIGHT-PATTERSON AFB OH 45433-7739	1

AFRL/HESC-TDC
2698 G STREET, BLDG 190
WRIGHT-PATTERSON AFB OH 45433-7604 — 1

ATTN: SMDC IM PL
US ARMY SPACE & MISSILE DEF CMD
P.O. BOX 1500
HUNTSVILLE AL 35807-3801 — 1

TECHNICAL LIBRARY D0274(PL-TS)
SPAWARSYSCEN
53560 HULL ST.
SAN DIEGO CA 92152-5001 — 1

COMMANDER, CODE 4TL000D
TECHNICAL LIBRARY, NAWC-WD
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100 — 1

CDR, US ARMY AVIATION & MISSILE CMD
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSAM-RD-OB-R, (DOCUMENTS)
REDSTONE ARSENAL AL 35898-5000 — 2

REPORT LIBRARY
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545 — 1

ATTN: D'BORAH HART
AVIATION BRANCH SVC 122.10
FOB10A, RM 931
800 INDEPENDENCE AVE, SW
WASHINGTON DC 20591 — 1

AFIWC/MSY
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016 — 1

ATTN: KAROLA M. YOURISON
SOFTWARE ENGINEERING INSTITUTE
4500 FIFTH AVENUE
PITTSBURGH PA 15213 — 1

USAF/AIR FORCE RESEARCH LABORATORY 1
AFRL/VSDOA(LIBRARY-BLDG 1103)
5 WRIGHT DRIVE
HANSCOM AFB MA 01731-3004

ATTN: EILEEN LADUKE/D460 1
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

OUSD(P)/DTSA/DUTD 1
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

SOFTWARE ENGR'G INST TECH LIBRARY 1
ATTN: MR DENNIS SMITH
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213-3890

USC-ISI 1
ATTN: DR ROBERT M. BALZER
4676 ADMIRALTY WAY
MARINA DEL REY CA 90292-6695

KESTREL INSTITUTE 1
ATTN: DR CORDELL GREEN
1801 PAGE MILL ROAD
PALO ALTO CA 94304

ROCHESTER INSTITUTE OF TECHNOLOGY 1
ATTN: PROF J. A. LASKY
1 LOMB MEMORIAL DRIVE
P.O. BOX 9887
ROCHESTER NY 14613-5708

AFIT/ENG 1
ATTN:TOM HARTRUM
WPAFB OH 45433-6583

THE MITRE CORPORATION 1
ATTN: MR EDWARD H. BENSLEY
BURLINGTON RD/MAIL STOP A350
BEDFORD MA 01730

ANDREW A. CHIEN 1
SAIC CHAIR PROF (SCI APL INT CORP)
USCD/CSE-APAM 4808
9500 GILMAN DRIVE, DEPT. 0114
LAJOLLA CA 92093-0114

HONEYWELL, INC. 1
ATTN: MR BERT HARRIS
FEDERAL SYSTEMS
7900 WESTPARK DRIVE
MCLEAN VA 22102

SOFTWARE ENGINEERING INSTITUTE 1
ATTN: MR. WILLIAM E. HEFLEY
CARNEGIE-MELLON UNIVERSITY
304 OAK GROVE CT
WEXFORD PA 15090

UNIVERSITY OF SOUTHERN CALIFORNIA 1
ATTN: DR. YIGAL ARENS
INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY/SUITE 1001
MARINA DEL REY CA 90292-6695

COLUMBIA UNIV/DEPT COMPUTER SCIENCE 1
ATTN: DR GAIL E. KAISER
450 COMPUTER SCIENCE BLDG
500 WEST 120TH STREET
NEW YORK NY 10027

AFIT/ENG 1
ATTN: DR GARY B. LAMONT
SCHOOL OF ENGINEERING
DEPT ELECTRICAL & COMPUTER ENGRG
WPAFB OH 45433-6583

NSA/OFC OF RESEARCH 1
ATTN: MS MARY ANNE OVERMAN
9800 SAVAGE ROAD
FT GEORGE G. MEADE MD 20755-6000

TEXAS INSTRUMENTS INCORPORATED 1
ATTN: DR DAVID L. WELLS
P.O. BOX 655474, MS 238
DALLAS TX 75265

KESTREL DEVELOPMENT CORPORATION 1
ATTN: DR RICHARD JULLIG
3260 HILLVIEW AVENUE
PALO ALTO CA 94304

DARPA/ITO 1
ATTN: DR KIRSTIE BELLMAN
3701 N FAIRFAX DRIVE
ARLINGTON VA 22203-1714

NASA/JOHNSON SPACE CENTER 1
ATTN: CHRIS CULBERT
MAIL CODE PT4
HOUSTON TX 77058

STERLING IMD INC. 1
KSC OPERATIONS
ATTN: MARK MAGINN
BEECHES TECHNICAL CAMPUS/RT 26 N.
ROME NY 13440

SCHLUMBERGER LABORATORY FOR 1
COMPUTER SCIENCE
ATTN: DR. GUILLERMO ARANGO
8311 NORTH FM620
AUSTIN, TX 78720

DECISION SYSTEMS DEPARTMENT 1
ATTN: PROF WALT SCACCHI
SCHOOL OF BUSINESS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CA 90089-1421

NATIONAL INSTITUTE OF STANDARDS 1
AND TECHNOLOGY
ATTN: CHRIS DABROWSKI
ROOM A266, BLDG 225
GAITHSBURG MD 20899

EXPERT SYSTEMS LABORATORY 1
ATTN: STEVEN H. SCHWARTZ
NYNEX SCIENCE & TECHNOLOGY
500 WESTCHESTER AVENUE
WHITE PLAINS NY 20604

NAVAL TRAINING SYSTEMS CENTER 1
ATTN: ROBERT BREAUX/CODE 252
12350 RESEARCH PARKWAY
ORLANDO FL 32826-3224

DR JOHN SALASIN 1
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DR BARRY BOEHM 1
DIR, USC CENTER FOR SW ENGINEERING
COMPUTER SCIENCE DEPT
UNIV OF SOUTHERN CALIFORNIA
LOS ANGELES CA 90089-0781

DR STEVE CROSS 1
CARNEGIE MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3891

DR MARK MAYBURY 1
MITRE CORPORATION
ADVANCED INFO SYS TECH; G041
BURLINGTON ROAD, M/S K-329
BEDFORD MA 01730

ISX 1
ATTN: MR. SCOTT FOUSE
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE,CA 91361

MR GARY EDMAPDS 1
ISX
433 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361

LEE ERMAN 1
CIMFLEX TEKNOWLEDGE
1010 EMBACADERO ROAD
P.O. BOX 10119
PALO ALTO CA 94303

DR. DAVE GUNNING 1
DARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DR. MICHAEL PITTARELLI 1
COMPUTER SCIENCE DEPART
SUNY INST OF TECH AT UTICA/ROME
P.O. BOX 3050
UTICA, NY 13504-3050

CAPRARO TECHNOLOGIES, INC 1
ATTN: GERARD CAPRARO
311 TURNER ST.
UTICA, NY 13501

USC/ISI
ATTN: BOB MCGREGOR
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292 1

SRI INTERNATIONAL
ATTN: ENRIQUE RUSPINI
333 RAVENSWOOD AVE
MENLO PARK, CA 94025 1

DARTMOUTH COLLEGE
ATTN: DANIELA RUS
DEPT OF COMPUTER SCIENCE
11 ROPE FERRY ROAD
HANOVER, NH 03755-3510 1

UNIVERSITY OF FLORIDA
ATTN: ERIC HANSON
CISE DEPT 456 CSE
GAINESVILLE, FL 32611-6120 1

CARNEGIE MELLON UNIVERSITY
ATTN: TOM MITCHELL
COMPUTER SCIENCE DEPARTMENT
PITTSBURGH, PA 15213-3890 1

UNIVERSITY OF ROCHESTER
ATTN: JAMES ALLEN
DEPARTMENT OF COMPUTER SCIENCE
ROCHESTER, NY 14627 1

MNIS-TEXTWISE LABS
ATTN: PARAIE SHERIDAN
DEY CENTENNIAL PLAZA 5TH FLOOR
SYRACUSE, NY 13502 1

WRIGHT STATE UNIVERSITY
ATTN: DR. BRUCE BERRA
DEPART OF COMPUTER SCIENCE & ENGIN
DAYTON, OHIO 45435-0001 1

UNIVERSITY OF FLORIDA
ATTN: SHARMA CHAKRAVARTHY
COMPUTER & INFOR SCIENCE DEPART
GAINESVILLE, FL 32622-6125 1

KESTREL INSTITUTE 1
ATTN: DAVID ESPINOSA
3260 HILLVIEW AVENUE
PALO ALTO, CA 94304

USC/INFORMATION SCIENCE INSTITUTE 1
ATTN: DR. CARL KESSELMAN
11474 ADMIRALTY WAY, SUITE 1001
MARINA DEL REY, CA 90292

MASSACHUSETTS INSTITUTE OF TECH 1
ATTN: DR. MICHAELE SIEGEL
SLOAN SCHOOL
77 MASSACHUSETTS AVENUE
CAMBRIDGE, MA 02139

USC/INFORMATION SCIENCE INSTITUTE 1
ATTN: DR. WILLIAM SWARTHOUT
11474 ADMIRALTY WAY, SUITE 1001
MARINA DEL REY, CA 90292

STANFORD UNIVERSITY 1
ATTN: DR. GIO WIEDERHOLD
857 SIERRA STREET
STANFORD
SANTA CLARA COUNTY, CA 94305-4125

SPAWARSYSCEN D44209 1
ATTN: LEAH WONG
53245 PATTERSON ROAD
SAN DIEGO, CA 92152-7151

SPAWARSYSCEN D4123 1
ATTN: LES ANDERSON
53560 HULL STREET
SAN DIEGO CA 92152-5001

GEORGE MASON UNIVERSITY 1
ATTN: SUSHIL JAJODIA
ISSE DEPT
FAIRFAX, VA 22030-4444

DIRNSA 1
ATTN: MICHAEL R. WARE
DOD, NSA/CSS (R23)
FT. GEORGE G. MEADE MD 20755-6000

DR. JIM RICHARDSON 1
3660 TECHNOLOGY DRIVE
MINNEAPOLIS, MN 55418

LOUISIANA STATE UNIVERSITY 1
COMPUTER SCIENCE DEPT
ATTN: DR. PETER CHEN
257 COATES HALL
BATON ROUGE, LA 70803

INSTITUTE OF TECH DEPT OF COMP SCI 1
ATTN: DR. JAIDEEP SRIVASTAVA
4-192 EE/CS
200 UNION ST SE
MINNEAPOLIS, MN 55455

GTE/BBN 1
ATTN: MAURICE M. MCNEIL
9655 GRANITE RIDGE DRIVE
SUITE 245
SAN DIEGO, CA 92123

UNIVERSITY OF FLORIDA 1
ATTN: DR. SHARMA CHAKRAVARTHY
E470 CSE BUILDING
GAINESVILLE, FL 32611-6125

AFRL/IFT 1
525 BROOKS ROAD
ROME, NY 13441-4505

AFRL/IFTM 1
525 BROOKS ROAD
ROME, NY 13441-4505

JEAN SCHOLTZ 1
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DR. ROGER CHEN 1
DEPT OF ELECT & COMPUTER ENGINR
SYRACUSE UNIVERSITY
SYRACUSE, NY 13244-1240

MISSION OF AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.